D206 Data Cleaning Performance Assessment

Ednalyn C. De Dios

For this performance assessment, I have chosen to use the Telco churn data set.

Part I. Research Question

A. Question or Decision

> ### *Are customers who had not set up an automatic payment method more likely to churn than those customers who had set up an automatic payment method?*

For this question, I will attempt to address it by looking at the "PaymentMethod" variable and making a new column that designates whether the payment method is automatic or not. I will call this variable "IsPaymentAutomatic." For example, if a record's "PaymentMethod" is electronic check, bank (automatic bank transfer), or credit card (automatic), I will classify the "IsPaymentAutomatic" as 1. Otherwise, 0 for everything else.

> ### *What factors contribute to customer churn?*

This second research question needs to be answered by using and considering all variables in the data set. I can extract feature importance by using xgboost or random forest, for example.

In [1]:
```
# setting the random seed for reproducibility
import random
random.seed(493)

import pandas as pd # for manipulating dataframes
import numpy as np # for numerical operations
import matplotlib.pyplot as plt # for visualization
from sklearn.impute import SimpleImputer # for handling missing values

# to print out all the outputs
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# set display options
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)
```

In [2]:

```python
# Read a csv file
df = pd.read_csv('churn_raw_data.csv', index_col=0)
```

In [3]:

```python
# Preview the data
df.head(5)
df.tail(5)
df.shape
```

Out[3]:

	CaseOrder	Customer_id	Interaction	City	State	County	Zip	Lat	
1	1	K409198	aa90260b-4141-4a24-8e36-b04ce1f4f77b	Point Baker	AK	Prince of Wales-Hyder	99927	56.25100	-13
2	2	S120509	fb76459f-c047-4a9d-8af9-e0f7d4ac2524	West Branch	MI	Ogemaw	48661	44.32893	-84
3	3	K191035	344d114c-3736-4be5-98f7-c72c281e2d35	Yamhill	OR	Yamhill	97148	45.35589	-12
4	4	D90850	abfa2b40-2d43-4994-b15a-989b8c79e311	Del Mar	CA	San Diego	92014	32.96687	-11
5	5	K662701	68a861fd-0d20-4e51-a587-8a90407ee574	Needville	TX	Fort Bend	77461	29.38012	-9

Out[3]:

	CaseOrder	Customer_id	Interaction	City	State	County	Zip	
9996	9996	M324793	45deb5a2-ae04-4518-bf0b-c82db8dbe4a4	Mount Holly	VT	Rutland	5758	43.4
9997	9997	D861732	6e96b921-0c09-4993-bbda-a1ac6411061a	Clarksville	TN	Montgomery	37042	36.5
9998	9998	I243405	e8307ddf-9a01-4fff-bc59-4742e03fd24f	Mobeetie	TX	Wheeler	79061	35.5
9999	9999	I641617	3775ccfc-0052-4107-81ae-9657f81ecdf3	Carrollton	GA	Carroll	30117	33.5
10000	10000	T38070	9de5fb6e-bd33-4995-aec8-f01d0172a499	Clarkesville	GA	Habersham	30523	34.7

Out[3]: (10000, 51)

B. Required variables

| Variable | Data Type | Description | Example | -------------------- | -------------------- | -------------------- | -------------------- | | **CaseOrder** | Integer | Designates the original order of the raw data file | 1-10000 | **Customer_id** | String | Unique identifier for customers | K409198, S120509, K191035 | **Interaction** | String | Unique identifiers related to customer transactions, technical support, and sign-ups | aa90260b-4141-4a24-8e36-b04ce1f4f77b | **City** | String | Customer's city of residence | Mobeetie | **State** | String | Customer's state of residence | TX | **County** | String | Customer's county of residence | Wheeler | **Zip** | Integer* | Customer's ZIP code of residence | 79061, 30117, 30523 | **Lat** | Float | GPS coordinates for lattitude of customer's residence | 34.70783 | **Lng** | Float | GPS coordinates for longitude of customer's residence | 83.53648 | **Population** | Integer | Population within a mile radius of customer's residence based on census data | **Area** | String | Type of area for custommer's residence | Rural, Urban, Suburban | **TimeZone** | String | Time zone of customer's residence | America/Chicago | **Job** | String | Job title of the customer | IT technical support officer | **Children** | Float* | Number of children in the customer's household | 3.0, 4.0, 1.0 | **Age**| Float* | Customner's age | 48.0, 39.0, 28.0 | **Education** | String | Highest degree completed by cusstomer | Regular High School Diploma | **Income** | Float | Customer's annual income | 55723.74, 16667.58, 28561.99 | **Marital** | String | Customer's marital status | Married,

Divorced, Never Married, Separated | **Gender** | String | Customer's gender | Female, Male, Prefer not to answer | **Churn** | String | Whether the customer discontinued service within the last month | Yes, No | **Outage_sec_perweek** | Float | Average number of seconds per week of system outages in customer's neighborhood | **Email** | Integer | Number of emails sent to the customer in the last year | 12, 9, 15 | **Contacts** | Integer | Number of times the customer contacted technical support | 2, 1 | **Yearly_equip_failure** | Integer | Number of times the customer's equipment failed and had top be reset or replaced in the past year | 0, 1 | **Techie** | String | Whether the customer thinks themselves as technically inclined | Yes, No | **Contract**| String | The contract term for the customer |Month-to-month, Two Year | **Port_modem** | String | Whether the customer has a portable modem | Yes, No | **Tablet** | String | Whether the customer owns a tablet | Yes, No | **InternetService** | String | The customer's type of internet service | DSL, Fiber Optic | **Phone** | String | Whether the customer has a phone service | Yes, No | **Multiple** | String | Whether the customer has multiple phone lines | Yes, No | **OnlineSecurity** | String | Whether the customer has an online security add-on | Yes, No | **OnlineBackup** | String | Whether the customer has an online backup add-on | Yes, No | **DeviceProtection** | String | Whether the customer has device protection add-on | Yes, No | **TechSupport** | String | Whether the customer has technical support add-on | Yes, No | **StreamingTV** | String | Whether the customer has streaming TV | Yes, No | **StreamingMovies** | String | Whether the customer has streaming movies | Yes, No | **PaperlessBilling** | String | Whether the customer has enrolled in paperless billing | Yes, No | **PaymentMethod** | String | Customer's method of payment | Electronic Check, Bank Transfer(automatic) | **Tenure** | Float | Number of months the customer has stayed with the provider | 68.19713, 61.04037, 71.09560 | **MonthlyCharge** | Float | Amount charged to the customer monthly | 159.8288, 168.2209, 218.3710 | **Bandwidth_GB_Year** | Float | Average amount of data used by the customer in GB within a year | 6511.253, 5857.586, 4159.306 | **Item1** | Integer | Rate of the importance of timely response as rated by the customer with 1 being the most important and 8 being the least important | 1-8 | **Item2** | Integer | Rate of the importance of timely fixes as rated by the customer with 1 being the most important and 8 being the least important | 1-8 | **Item3** | Integer | Rate of the importance of timely replacements as rated by the customer with 1 being the most important and 8 being the least important | 1-8 | **Item4** | Integer | Rate of the importance of reliabilitye as rated by the customer with 1 being the most important and 8 being the least important | 1-8 | **Item5** | Integer | Rate of the importance of optionse as rated by the customer with 1 being the most important and 8 being the least important | 1-8 | **Item6** | Integer | Rate of the importance of respectful responsee as rated by the customer with 1 being the most important and 8 being the least important | 1-8 | **Item7** | Integer | Rate of the importance of courteous exchange as rated by the customer with 1 being the most important and 8 being the least important | 1-8 | **Item8** | Integer | Rate of the importance of evidence of active listening as rated by the customer with 1 being the most important and 8 being the least important | 1-8

* Wrong data type, needs to be transformed (typecasted)

Part II. Data Cleaning Plan

C1. Plan to Assess the Quality of Data

- Look for duplicates

> The duplicated() method will be used to find out if any duplicated rows exist in the data set. Then, the .shape attribute will be utilized to determine how many rows (if any) are duplicated.

- Find missing values

> df.isnull().sum() will be used to count the number of true NaNs in the dataset for each column. ((df == 'nan') | (df == 'NaN')).sum() will also be used to catch any literal "nan" or "NaN" values in the dataset for each column. The results will be presented in tabular format for readability.

- Examine data types for correctness

> I will use the .info() method to take note of the data types for each feature in the data set. Then, I will examine each one and use domain knowledge to verify if they are assigned the correct data type or not.

- Check for outliers

> I will utilize a function to loop through each outliers. The function will use boxplots to visually examine the outliers and print outs of values outside the upper and lower bound to identify those outliers.

C2. Justification of Approach to Assess the Quality of Data

The first step in my approach to assess the quality of the data is to look for duplicates. This is an important step because ignoring duplicates can skew the distribution of the data set and would result in incorrect visualizations like histograms.

The second step is to find missing values in the data set. This is a crucial step because missing data introduces bias into models. Missing data in the data set can also reduce the statistical power of analysis conducted on the data set.

The third step is to examine the data types for correctness. Doing so shall avoid errors stemming from incorrect data types and mismatched values.

The last step is to check for any outliers. This is necessary if the aforementioned outliers cause unnecessary skewness in the data distribution.

C3. Justification of Tools Used to Assess the Quality of Data

I will use Python to assess the quality of the data set. Using a few libraries and packages such as pandas and matplotlib will allow me to examine and review the untidiness in the data. For example, pandas has an isnull() function that can be used to filter rows in the dataframe that has null values. Moreover, visualization packages like matplotlib allow me to eyeball any outliers in the data and give me a ballpark value to input so that I can filter and remove those outliers.

C4. Annotated Code Used to Assess the Quality of Data

- [x] Look for duplicates
- [x] Find missing values
- [x] Examine data types for correctness
- [x] Check for outliers

Look for duplicates

In [4]:
```
# Select rows that are duplicated based on all columns. Any records after the first
dup = df[df.duplicated()]

# Find out how many rows are duplicated
dup.shape
```

Out[4]:
```
(0, 51)
```

Find missing values

In [5]:
```
def show_missing(df):
    """
    Takes a dataframe and returns a dataframe with stats
    on missing and null values with their percentages.
    """
    null_count = df.isnull().sum()
    null_percentage = (null_count / df.shape[0]) * 100
    empty_count = pd.Series(((df == ' ') | (df == '')).sum())
    empty_percentage = (empty_count / df.shape[0]) * 100
    nan_count = pd.Series(((df == 'nan') | (df == 'NaN')).sum())
    nan_percentage = (nan_count / df.shape[0]) * 100
    dfx = pd.DataFrame({'num_missing': null_count, 'missing_percentage': null_perce
                        'num_empty': empty_count, 'empty_percentage': empty_percen
                        'nan_count': nan_count, 'nan_percentage': nan_percentage})
    return dfx

show_missing(df)
```

Out[5]:

	num_missing	missing_percentage	num_empty	empty_percentage	na
CaseOrder	0	0.00	0	0.0	
Customer_id	0	0.00	0	0.0	
Interaction	0	0.00	0	0.0	
City	0	0.00	0	0.0	
State	0	0.00	0	0.0	
County	0	0.00	0	0.0	
Zip	0	0.00	0	0.0	
Lat	0	0.00	0	0.0	
Lng	0	0.00	0	0.0	
Population	0	0.00	0	0.0	
Area	0	0.00	0	0.0	
Timezone	0	0.00	0	0.0	
Job	0	0.00	0	0.0	
Children	2495	24.95	0	0.0	
Age	2475	24.75	0	0.0	
Education	0	0.00	0	0.0	
Employment	0	0.00	0	0.0	
Income	2490	24.90	0	0.0	
Marital	0	0.00	0	0.0	
Gender	0	0.00	0	0.0	
Churn	0	0.00	0	0.0	
Outage_sec_perweek	0	0.00	0	0.0	
Email	0	0.00	0	0.0	
Contacts	0	0.00	0	0.0	
Yearly_equip_failure	0	0.00	0	0.0	
Techie	2477	24.77	0	0.0	
Contract	0	0.00	0	0.0	
Port_modem	0	0.00	0	0.0	
Tablet	0	0.00	0	0.0	
InternetService	2129	21.29	0	0.0	

	num_missing	missing_percentage	num_empty	empty_percentage	na
Phone	1026	10.26	0	0.0	
Multiple	0	0.00	0	0.0	
OnlineSecurity	0	0.00	0	0.0	
OnlineBackup	0	0.00	0	0.0	
DeviceProtection	0	0.00	0	0.0	
TechSupport	991	9.91	0	0.0	
StreamingTV	0	0.00	0	0.0	
StreamingMovies	0	0.00	0	0.0	
PaperlessBilling	0	0.00	0	0.0	
PaymentMethod	0	0.00	0	0.0	
Tenure	931	9.31	0	0.0	
MonthlyCharge	0	0.00	0	0.0	
Bandwidth_GB_Year	1021	10.21	0	0.0	
item1	0	0.00	0	0.0	
item2	0	0.00	0	0.0	
item3	0	0.00	0	0.0	
item4	0	0.00	0	0.0	
item5	0	0.00	0	0.0	
item6	0	0.00	0	0.0	
item7	0	0.00	0	0.0	
item8	0	0.00	0	0.0	

Examine datatypes for correctness

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10000 entries, 1 to 10000
Data columns (total 51 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CaseOrder             10000 non-null  int64  
 1   Customer_id           10000 non-null  object 
 2   Interaction           10000 non-null  object 
 3   City                  10000 non-null  object 
 4   State                 10000 non-null  object 
 5   County                10000 non-null  object 
 6   Zip                   10000 non-null  int64  
 7   Lat                   10000 non-null  float64
 8   Lng                   10000 non-null  float64
 9   Population            10000 non-null  int64  
 10  Area                  10000 non-null  object 
 11  Timezone              10000 non-null  object 
 12  Job                   10000 non-null  object 
 13  Children              7505 non-null   float64
 14  Age                   7525 non-null   float64
 15  Education             10000 non-null  object 
 16  Employment            10000 non-null  object 
 17  Income                7510 non-null   float64
 18  Marital               10000 non-null  object 
 19  Gender                10000 non-null  object 
 20  Churn                 10000 non-null  object 
 21  Outage_sec_perweek    10000 non-null  float64
 22  Email                 10000 non-null  int64  
 23  Contacts              10000 non-null  int64  
 24  Yearly_equip_failure  10000 non-null  int64  
 25  Techie                7523 non-null   object 
 26  Contract              10000 non-null  object 
 27  Port_modem            10000 non-null  object 
 28  Tablet                10000 non-null  object 
 29  InternetService       7871 non-null   object 
 30  Phone                 8974 non-null   object 
 31  Multiple              10000 non-null  object 
 32  OnlineSecurity        10000 non-null  object 
 33  OnlineBackup          10000 non-null  object 
 34  DeviceProtection      10000 non-null  object 
 35  TechSupport           9009 non-null   object 
 36  StreamingTV           10000 non-null  object 
 37  StreamingMovies       10000 non-null  object 
 38  PaperlessBilling      10000 non-null  object 
 39  PaymentMethod         10000 non-null  object 
 40  Tenure                9069 non-null   float64
 41  MonthlyCharge         10000 non-null  float64
 42  Bandwidth_GB_Year     8979 non-null   float64
 43  item1                 10000 non-null  int64  
 44  item2                 10000 non-null  int64  
 45  item3                 10000 non-null  int64  
 46  item4                 10000 non-null  int64  
 47  item5                 10000 non-null  int64  
 48  item6                 10000 non-null  int64  
 49  item7                 10000 non-null  int64  
 50  item8                 10000 non-null  int64  
```

```
dtypes: float64(9), int64(14), object(28)
memory usage: 4.0+ MB
```

Check for outliers

In [7]:
```python
def show_outliers(column):
    """
    Takes a column and displays a boxplot, iqr, upper and
    lower bound, along with a listing of the outliers.
    """
    plt.boxplot(df[column])
    fig = plt.figure(figsize =(10, 7))

    # finding the 1st quartile
    q1 = np.quantile(df[column], 0.25)

    # finding the 3rd quartile
    q3 = np.quantile(df[column], 0.75)
    med = np.median(df[column])

    # finding the iqr region
    iqr = q3-q1

    # finding upper and lower whiskers
    upper_bound = q3+(1.5*iqr)
    lower_bound = q1-(1.5*iqr)
    print(iqr, upper_bound, lower_bound)

    outliers = df[column][(df[column] <= lower_bound) | (df[column] >= upper_bound)
    print('---------------------------------------------------------------------')
    print(column)
    print('---------------------------------------------------------------------')
    print('The following are the outliers in the boxplot\n{}'.format(outliers))
    print('\n\n')
```

In [8]:
```python
# assemble a list of column names that are good candidates for outliers
outlier_columns = ['Population',
                   'Outage_sec_perweek',
                   'Email',
                   'Contacts',
                   'Yearly_equip_failure',
                   'MonthlyCharge'
                  ]

# loop over the list of column names
for col in outlier_columns:
    show_outliers(col)
```

```
12430.0 31813.0 -17907.0
---------------------------------------------------------------------
Population
---------------------------------------------------------------------
The following are the outliers in the boxplot
12       33372
17       50079
30       52484
45       35743
52       39649
57       46869
58       58431
67       38476
75       34359
86       35279
88       32203
91       55519
101      55122
103      31927
111      41733
121      37711
124      31859
142      43123
157      47732
158      86926
164      32653
172      43714
204      90517
213      62430
216      44451
218      33649
232      47974
241      57344
242      39035
257      34993
258      36260
260      41839
263      32603
276      48969
286      32084
292      61045
315      38579
324      34669
352      35345
353      41155
361      34460
373      51767
380      32929
385      46064
386      32525
395      31845
427      35100
433      41863
437      40305
442      39616
443      88349
```

446	56959
465	34928
469	49612
499	34488
529	48990
530	43597
556	74971
559	36282
578	63318
588	32539
590	46581
593	61572
599	51069
605	33287
622	45193
645	54150
647	44647
656	39641
663	53140
739	45995
741	52299
745	33484
748	36817
772	46920
775	39384
780	57511
802	31819
822	33168
830	76973
837	61509
838	35349
852	40337
856	35603
870	51582
891	33725
898	44826
907	55652
970	37464
986	39624
991	57642
1002	47367
1044	47645
1086	40019
1097	35102
1108	46775
1114	50288
1132	60461
1151	34784
1157	38753
1160	57955
1163	41000
1174	79276
1187	37540
1192	66531
1196	39088
1212	87240

1219	35824
1225	32195
1234	37774
1261	38885
1265	37976
1274	39381
1305	36202
1323	32825
1324	48863
1328	53364
1343	51447
1348	56765
1357	69142
1365	33049
1399	89075
1430	46286
1461	57800
1466	42532
1529	47493
1532	58129
1554	42582
1563	40770
1567	45487
1578	38436
1579	32351
1586	33103
1602	34945
1624	56284
1633	38443
1643	35457
1689	46589
1691	39478
1699	36923
1715	47493
1734	35763
1748	34308
1771	64417
1775	35591
1776	98660
1784	32000
1786	53431
1792	59191
1815	64417
1821	43123
1827	40041
1838	41468
1841	59586
1844	35151
1848	35389
1851	41235
1889	46344
1892	49397
1894	94395
1903	48133
1917	32944
1929	35376

1941	39568
1942	33225
1956	35414
1958	42227
1959	47459
1961	39694
1966	48008
1969	35771
1977	39842
2006	34312
2020	35373
2023	51369
2031	36184
2033	31990
2048	54829
2062	43116
2065	36172
2075	44139
2080	39234
2081	44455
2094	35465
2102	42116
2106	46675
2125	40290
2135	37122
2139	50137
2153	38787
2161	43881
2164	54621
2176	35349
2194	51045
2202	41468
2212	81789
2217	37358
2218	42161
2230	34237
2248	35373
2256	39065
2273	39273
2275	72332
2281	47227
2283	40860
2295	49350
2297	55222
2298	44740
2307	47791
2309	40820
2311	72592
2317	34119
2325	38932
2343	51706
2366	41053
2392	37056
2395	42116
2401	36062
2403	94512

2414	76819
2416	32653
2418	33380
2433	58123
2474	54023
2477	33875
2487	34639
2516	54023
2537	39850
2538	32572
2571	35102
2579	36986
2581	38865
2584	56308
2612	62429
2613	39227
2627	37375
2647	32801
2682	34791
2693	49344
2696	40005
2702	46589
2716	44249
2720	47974
2737	41454
2738	40828
2753	49019
2764	52454
2801	34305
2807	33960
2828	46545
2869	51667
2871	41149
2904	38753
2907	34941
2912	62131
2930	46327
2934	34381
2940	39344
2953	71763
2955	40041
2960	56446
2980	41861
2986	32388
2990	37035
2999	41523
3002	50869
3019	54190
3026	37067
3028	52078
3031	38835
3042	40543
3054	80555
3062	36753
3063	36335
3067	41818

3075	35279
3078	31883
3086	40649
3093	36613
3098	47309
3100	39658
3114	35061
3128	61045
3145	65123
3153	34119
3156	32135
3159	57181
3165	33240
3173	54081
3177	32363
3186	40063
3195	60408
3201	55215
3219	76177
3224	40340
3228	36531
3239	38476
3244	34519
3255	32066
3259	36613
3264	41407
3276	38524
3279	58705
3282	40866
3291	40825
3292	66053
3293	35087
3307	45763
3323	59556
3329	53098
3330	56406
3338	74971
3361	32211
3370	55623
3372	52669
3379	38620
3382	38334
3393	74601
3402	51447
3405	32564
3426	34056
3428	80489
3453	57955
3456	62382
3458	37201
3468	56790
3473	40857
3481	49271
3487	32315
3496	32688
3505	52371

3506	34496
3530	35317
3542	38073
3565	36184
3572	44328
3579	39090
3596	33685
3610	39859
3623	36455
3624	46875
3630	36312
3634	33328
3638	40837
3649	68676
3650	37878
3656	45763
3683	32901
3695	40604
3696	34918
3704	46276
3705	36265
3722	37639
3732	36681
3738	48854
3764	44239
3773	35489
3774	39309
3783	35575
3808	36224
3813	35465
3814	36047
3823	44145
3830	34675
3832	46041
3842	46418
3862	42041
3866	32069
3875	49836
3882	52967
3916	34470
3917	32739
3918	40447
3920	51275
3933	41387
3949	63393
3952	34950
3953	54464
3965	42059
3970	36409
3981	48008
4004	57658
4012	33960
4018	33420
4020	34191
4037	36489
4039	47459

4043	41510
4055	35798
4059	36539
4070	32959
4071	49787
4076	50640
4083	35952
4091	79996
4092	32388
4106	45596
4126	33786
4144	73671
4150	33760
4151	36792
4178	38272
4180	33168
4188	45763
4199	45910
4200	39757
4244	43795
4246	40170
4257	44460
4262	53552
4263	49019
4264	48351
4269	53364
4279	63659
4296	36923
4303	39523
4305	32029
4310	56642
4327	35523
4333	38028
4334	34624
4344	48833
4350	90675
4359	35157
4371	39786
4376	62491
4388	39376
4397	40336
4404	35142
4411	73671
4431	38016
4453	69589
4464	48099
4467	42650
4479	33014
4509	37360
4515	47757
4516	57886
4520	60033
4532	44103
4537	37042
4572	48438
4605	55857

4637	57775
4658	34135
4667	38909
4673	42979
4680	38748
4712	34313
4718	52688
4730	68300
4731	39698
4743	60944
4751	34315
4753	50328
4778	39642
4780	46623
4786	34390
4797	66056
4810	38833
4825	43714
4831	33965
4839	38733
4848	48905
4855	38620
4859	43736
4865	62135
4880	45158
4900	38517
4906	41468
4910	39859
4921	52254
4945	55352
4957	63911
4963	35345
4973	63465
4981	31848
4996	56308
5015	32275
5038	63425
5042	40622
5052	75635
5079	83615
5084	35575
5091	38388
5109	41563
5119	34365
5127	35840
5140	38547
5172	59609
5184	68128
5188	58673
5191	38267
5197	50972
5207	48386
5216	39012
5226	40752
5246	61850
5259	41454

5263	71360
5274	46024
5278	35950
5281	33442
5289	57181
5292	38376
5298	36441
5302	58160
5308	37204
5322	49657
5371	37183
5373	60270
5403	44139
5411	38579
5416	37646
5423	48386
5452	34100
5455	38013
5458	67053
5462	47346
5496	37080
5501	83179
5507	38424
5508	34506
5513	34355
5552	80489
5558	49790
5566	51003
5587	41961
5589	50869
5604	44468
5634	36441
5645	45487
5651	32525
5655	59009
5656	53886
5659	41204
5671	35436
5672	63218
5692	49158
5706	36138
5707	49694
5714	37476
5716	35087
5721	34639
5724	45694
5726	42991
5729	51767
5738	66583
5752	39731
5754	47309
5775	54601
5803	38553
5809	51426
5815	65347
5831	62274

5855	46333
5876	32681
5891	34381
5899	88344
5900	63884
5901	56641
5902	59482
5906	82257
5909	38659
5934	32031
5951	46581
5958	47710
5974	35702
5983	36962
5991	36293
6021	39391
6032	49630
6059	34784
6075	41018
6083	49894
6089	53364
6117	73671
6136	37906
6152	32096
6154	51447
6158	33626
6165	70614
6169	80264
6172	57306
6180	38363
6181	40385
6223	39440
6226	66053
6241	36531
6244	40434
6245	40778
6248	44968
6254	33612
6283	37641
6289	102433
6321	35151
6332	34794
6350	34642
6359	33031
6360	52322
6376	34082
6381	40441
6388	32901
6401	39062
6404	71878
6409	48770
6434	42524
6465	87509
6496	35101
6501	54101
6513	74172

6524	48713
6525	51275
6527	45298
6534	44760
6544	34488
6548	37527
6562	32526
6565	38476
6570	43149
6572	32471
6588	32264
6591	35681
6597	50386
6600	37207
6602	43112
6606	35923
6608	35999
6611	96575
6631	52939
6646	40439
6655	38957
6658	40425
6661	35107
6664	34545
6671	39078
6678	37533
6686	49870
6694	46299
6730	32035
6743	73891
6745	50137
6752	37358
6776	32737
6777	37234
6797	34460
6808	42059
6845	40336
6853	35462
6862	43112
6874	36731
6875	55737
6882	49664
6886	33769
6899	86421
6916	35840
6921	42442
6934	52322
6947	57344
6954	61258
6974	42042
6977	53866
7005	32489
7013	32269
7021	35028
7043	76811
7083	58167

7084	38406
7099	65853
7101	32929
7103	46087
7109	40337
7111	39698
7118	43232
7134	39600
7141	49877
7143	58439
7146	50992
7169	53582
7196	31860
7197	40685
7200	31816
7204	40622
7244	58114
7264	52484
7276	54413
7293	35118
7328	46041
7335	34642
7343	48382
7351	42614
7355	39523
7361	48099
7366	39088
7384	61495
7412	37080
7416	34276
7419	33240
7429	52802
7431	47067
7436	33235
7442	94512
7444	40684
7445	72332
7449	56838
7455	86703
7494	62429
7496	38839
7503	33965
7504	51117
7512	56811
7532	41107
7534	48833
7558	33000
7561	53879
7579	78900
7582	72061
7588	52254
7592	38177
7612	56765
7639	41839
7649	65665
7652	51097

7654	37492
7672	36423
7684	52922
7688	45652
7700	55937
7716	33960
7731	38334
7732	35270
7740	37080
7758	37639
7765	34519
7766	36962
7793	41392
7797	56233
7806	38383
7832	72479
7839	46327
7849	38335
7850	48775
7862	45158
7892	51082
7900	33551
7910	79010
7914	32019
7922	40248
7923	62916
7949	41151
7961	33661
7963	40778
7966	42614
7974	34781
7976	38321
8002	32257
8007	61860
8026	61818
8034	42741
8036	41014
8038	42463
8041	46647
8049	48382
8051	45885
8061	66531
8080	86811
8084	44641
8087	58797
8105	31859
8107	42522
8120	70199
8131	96436
8140	111850
8148	39221
8185	35124
8187	51003
8193	42116
8206	43208
8208	42023

8214	38072
8218	53142
8221	62625
8249	41563
8251	36959
8270	37080
8276	54190
8303	49907
8321	103732
8328	41276
8338	74413
8347	43461
8355	42812
8358	33953
8377	32568
8381	55652
8385	43801
8388	60699
8392	63894
8393	46904
8403	32259
8419	32013
8424	35317
8443	44300
8468	44759
8474	35573
8475	45688
8484	36414
8498	35643
8503	77342
8507	42910
8520	43078
8545	42614
8555	53111
8566	44750
8586	59324
8596	32812
8603	39962
8622	35161
8644	49487
8655	44233
8660	33901
8676	33015
8682	60191
8715	34962
8718	69413
8730	32080
8731	35230
8736	40335
8742	38597
8750	36792
8751	49694
8753	73778
8758	60476
8766	38568
8768	33287

8802	36550
8803	36711
8814	39043
8839	40770
8850	37377
8890	83782
8902	38659
8905	53841
8908	33735
8928	59607
8934	32204
8939	44352
8945	42868
8947	33723
8948	80143
8952	39478
9003	65355
9014	40778
9044	35643
9051	40131
9057	75306
9059	62043
9065	41901
9066	60781
9083	36245
9089	52394
9104	53686
9112	56641
9118	46995
9119	34744
9149	41480
9175	59637
9177	41600
9197	32870
9199	56959
9202	41026
9206	65335
9211	46418
9214	41743
9229	63894
9237	40770
9242	44474
9251	46215
9256	54101
9328	60254
9335	55210
9338	40005
9345	41600
9357	36724
9390	43551
9422	35240
9427	39994
9451	54780
9463	32117
9475	48969
9494	57226

```
9499     34639
9501     33435
9505     42011
9508     46513
9510     41665
9512     46890
9551     35151
9555     42059
9577     41743
9579     47346
9617     69589
9620     35799
9624     53266
9633     43209
9642     42272
9648     54540
9662     35885
9675     38659
9679     35913
9695     36613
9729     54507
9749     32269
9762     31846
9785     33649
9786     46890
9797     32261
9803     37727
9818     43633
9822     40014
9839     43123
9854     34271
9856     45193
9859     48710
9863     32005
9869     35054
9877     32603
9891     32477
9899     39065
9906     54413
9912     49344
9926     40830
9937     38703
9980     32425
9988     87509
9997     77168
9999     35575
Name: Population, dtype: int64



4.433281622499997 19.137566056249995 1.404439566250005
--------------------------------------------------------------------
Outage_sec_perweek
--------------------------------------------------------------------
The following are the outliers in the boxplot
29      43.927052
```

37	44.725202
41	38.905335
62	39.883903
73	32.030945
131	39.696851
135	33.930542
177	38.842394
193	40.454053
205	40.904448
215	46.511607
228	40.378260
244	34.949063
289	44.708292
292	38.173986
302	39.926856
324	39.659140
347	42.317092
409	40.605172
411	39.901392
429	36.001063
433	38.493081
445	42.337606
451	40.993629
453	34.875483
466	38.258236
468	46.054249
503	42.551748
528	19.547280
598	37.346614
607	39.483708
625	39.209050
628	36.402870
643	38.397254
646	40.828059
666	35.505764
677	39.910978
686	33.021440
730	36.449434
734	37.577460
747	39.447940
799	39.846615
808	1.272758
811	41.768242
826	36.093439
851	46.021694
900	36.039752
904	33.660545
909	0.169351
928	37.018121
948	39.865476
986	38.199337
991	36.165608
998	37.823087
1023	19.272782
1045	36.556295
1047	36.277985

1062	36.119341
1073	39.069884
1077	34.903839
1081	41.496063
1086	39.136730
1098	44.578211
1118	33.549958
1128	39.625600
1167	35.607179
1173	39.441257
1175	35.443089
1185	44.390831
1192	41.487840
1217	37.233410
1236	43.153525
1273	37.835642
1294	31.033709
1295	45.900879
1307	19.242988
1317	34.108057
1388	39.288649
1424	35.207775
1431	35.433362
1559	37.201166
1574	43.288867
1584	40.485220
1594	36.868388
1602	37.692137
1610	40.168008
1623	42.170200
1640	43.137367
1642	38.737032
1644	37.788092
1726	37.901175
1730	35.166553
1780	46.641806
1799	41.304041
1819	35.217898
1826	36.275276
1857	40.613550
1868	39.420560
1905	-1.195428
1945	36.195472
1950	38.433088
1968	39.559878
1977	35.641460
1992	38.964319
1998	-0.339214
2017	35.539169
2022	41.454245
2023	38.377500
2038	39.747254
2077	42.991946
2174	41.217995
2203	39.242788
2208	37.609703

2221	38.700744
2225	39.442134
2251	19.572184
2256	38.888220
2269	19.300773
2290	41.078366
2304	37.466028
2319	35.781665
2328	36.508732
2404	39.880887
2409	40.472763
2426	38.358851
2433	40.083427
2449	39.830199
2457	39.206335
2471	36.553698
2478	38.350448
2515	37.211238
2529	35.528139
2536	35.975346
2552	42.280812
2582	42.482471
2589	40.784444
2619	38.014990
2621	40.565835
2635	40.350537
2656	41.015078
2662	31.693495
2687	39.203043
2719	38.426687
2741	36.788030
2773	36.551329
2796	34.798536
2805	38.283063
2810	38.103795
2826	38.750850
2861	36.207450
2884	35.318656
2887	35.501854
2896	41.497709
2899	1.201828
2917	41.302953
2926	38.014098
2934	36.100384
2938	37.216209
2948	38.888741
2951	39.436941
2961	33.920410
2962	39.894569
2971	38.603160
2985	0.840953
3017	42.820450
3019	43.498160
3069	40.062410
3070	-0.206145
3129	44.061880

3201	37.086330
3229	36.488010
3238	38.680050
3256	40.026060
3262	39.946730
3284	35.240220
3333	37.595920
3391	38.234480
3415	39.799010
3440	44.938000
3467	35.432570
3482	36.538380
3518	36.840580
3529	38.083110
3536	41.364710
3548	39.518520
3571	37.458300
3604	41.389590
3630	-0.152845
3647	40.289280
3691	37.617550
3697	35.427470
3700	38.969780
3703	41.742910
3747	40.421970
3760	1.263475
3782	36.443020
3797	40.011400
3802	41.203740
3809	41.275350
3825	37.516390
3840	41.331290
3845	41.698560
3882	36.963630
3892	36.341390
3894	42.461670
3899	42.751680
3924	39.649900
3957	40.692210
3996	39.518060
4026	39.567050
4052	34.905200
4121	33.898240
4133	41.928960
4143	39.614250
4161	38.055500
4168	-1.348571
4185	-0.352431
4191	19.883050
4216	38.868650
4247	41.550810
4267	38.732970
4284	38.807350
4313	1.248096
4330	41.247070
4343	42.784810

4400	40.646740
4406	37.441440
4426	38.592460
4428	-1.099934
4439	36.512760
4441	39.830390
4449	37.466800
4453	41.079760
4460	38.410790
4465	34.129400
4479	35.860220
4484	40.312860
4502	39.497210
4505	38.172120
4508	38.978850
4530	45.026000
4563	19.741330
4587	39.478860
4590	41.749280
4595	36.000460
4609	36.641950
4611	38.296870
4621	47.049280
4667	40.634280
4698	0.278712
4701	38.814030
4740	35.757500
4747	34.004630
4770	37.692510
4810	39.491750
4816	41.230360
4824	38.599830
4829	38.986870
4848	40.098810
4850	38.040370
4860	39.935760
4884	40.788640
5006	44.792460
5029	45.067050
5043	39.998190
5078	44.601890
5086	37.374790
5128	34.721600
5134	41.514330
5145	39.032000
5155	35.876770
5157	35.034990
5158	42.953310
5161	37.453940
5164	38.736680
5181	38.923220
5183	33.632670
5185	44.448780
5192	40.457060
5222	41.403140
5268	38.108980

5269	40.921350
5283	36.939180
5315	40.267910
5331	40.109520
5343	34.773460
5358	41.571670
5364	19.191610
5376	39.991160
5393	21.216190
5406	41.085980
5407	38.405000
5465	35.645300
5468	44.177030
5550	45.914470
5558	39.317530
5565	40.685440
5641	35.999020
5672	39.188070
5681	20.675440
5733	39.000960
5749	43.239040
5761	34.740530
5787	42.603940
5810	44.474020
5855	41.167610
5868	35.932720
5924	33.753840
5926	38.720530
5946	42.616810
6025	42.155570
6041	43.502540
6042	41.387390
6045	37.705150
6053	35.447100
6081	35.770880
6094	-0.787115
6128	41.570640
6130	37.209960
6133	40.367000
6134	31.486710
6158	40.176930
6176	36.126260
6179	36.970630
6189	41.703780
6191	37.123750
6246	45.240000
6266	37.347740
6281	38.940480
6292	41.625330
6359	37.355510
6420	36.754860
6426	39.217730
6434	45.758110
6464	-0.144644
6472	1.109474
6504	41.299510

6562	44.212670
6578	-0.527396
6596	39.101680
6619	37.291490
6669	47.027660
6670	40.314560
6690	36.527020
6693	34.716330
6709	42.651510
6721	35.936500
6734	36.920560
6762	43.927260
6765	42.990800
6767	42.359400
6777	36.400250
6790	35.466640
6805	44.233840
6829	39.860300
6852	19.528250
6870	42.828890
6875	37.743410
6877	40.561070
6888	36.728670
6919	34.200170
6942	36.514430
6947	40.646340
7002	39.851430
7017	35.998730
7029	35.669970
7060	39.667170
7071	0.359073
7077	42.541550
7085	38.204750
7104	35.774010
7135	39.915000
7148	40.009140
7153	40.518210
7195	38.097460
7208	41.964820
7237	38.478510
7252	42.100280
7268	38.608440
7293	34.517950
7316	43.473780
7340	0.113821
7390	0.994552
7399	40.674390
7408	34.528460
7416	40.105510
7419	38.740570
7460	39.106370
7468	36.732820
7482	40.398540
7512	32.608180
7524	40.596010
7535	34.488500

7575	42.147650
7656	40.790660
7666	43.222620
7671	45.035380
7673	30.039370
7736	37.987860
7748	41.035150
7759	38.889160
7784	38.159210
7792	36.403090
7798	40.601330
7821	33.695680
7906	39.315210
7918	39.430650
7935	33.994570
7939	39.116230
7947	39.632950
7953	34.832550
7954	19.868070
7968	37.878610
7989	38.789470
7994	36.307380
8053	41.510560
8076	38.188970
8085	37.935450
8093	42.288360
8097	40.640340
8111	40.197280
8125	39.344890
8157	38.519680
8168	35.667440
8175	38.430100
8181	0.915846
8192	0.852520
8195	-0.214328
8219	38.173910
8251	36.028450
8274	39.485670
8307	36.773830
8379	37.866930
8480	37.224410
8500	41.505890
8506	36.641880
8535	42.038660
8540	20.729480
8558	34.331650
8562	37.138930
8583	34.159900
8606	35.314720
8610	41.147340
8620	38.666560
8657	39.968150
8675	40.461730
8695	40.079540
8724	36.668100
8745	33.640110

8756	39.016490
8758	42.967380
8775	1.049154
8800	43.141440
8824	30.969870
8841	41.584630
8878	44.520640
8905	37.664830
8915	40.624180
8933	33.316610
8961	39.243880
8966	35.328410
8972	34.714260
9021	38.639810
9033	37.917830
9039	41.878320
9059	44.004130
9068	41.522860
9074	39.703320
9099	35.846890
9101	38.864500
9102	42.781430
9128	39.894560
9136	39.704610
9180	40.662600
9183	35.569610
9184	32.441070
9220	41.069520
9279	34.921680
9285	38.441660
9286	34.850480
9288	37.591900
9292	38.311240
9319	44.496010
9320	40.752650
9328	37.698460
9332	37.756080
9337	38.630110
9362	36.793210
9375	40.461650
9377	39.017700
9390	40.679160
9403	0.683623
9432	35.615170
9433	41.533570
9435	37.031560
9443	41.198570
9460	36.821520
9490	38.089020
9504	40.220160
9509	43.276320
9521	39.703370
9522	36.928400
9535	34.134010
9550	37.761470
9572	35.714070

```
9586    39.104720
9623    37.031640
9629    39.413930
9640    36.673400
9655    32.829480
9662    43.865160
9666    37.168540
9670    40.150420
9677    38.730220
9685    34.099650
9708    35.272380
9709    37.614990
9721    42.915150
9734    38.945510
9746    40.264350
9763    36.936770
9775    39.998330
9795    31.518410
9813    40.428570
9837    32.581260
9851    37.531360
9852    38.718870
9853    38.441150
9861    40.343850
9867    39.106320
9883    39.395180
9892    33.867310
9894    39.494660
9895    44.499730
9896    40.684860
9908    38.524730
9946    39.337010
9951    40.974290
9981    30.732980
Name: Outage_sec_perweek, dtype: float64


4.0 20.0 4.0
----------------------------------------------------------------------
Email
----------------------------------------------------------------------
The following are the outliers in the boxplot
9       20
93       3
259     21
262     20
426     20
488      4
688     20
796      2
800      4
929      4
1009    20
1115    20
1153     2
```

1177	20
1382	1
1389	20
1400	2
1417	4
1430	20
1474	23
1747	22
1809	21
1948	20
2196	3
2276	3
2329	21
2383	20
2441	4
2500	4
2807	21
2933	20
3033	20
3037	20
3066	4
3189	21
3267	21
3475	20
3594	3
3732	4
3797	20
4047	20
4291	20
4380	4
4612	4
4818	3
5016	4
5062	20
5208	20
5234	20
5437	20
5461	20
5559	21
5598	4
5617	4
5625	4
5661	20
5845	3
5886	20
6179	20
6193	4
6298	20
6321	1
6331	3
6389	4
6400	4
6403	4
6523	4
6545	4
6626	4

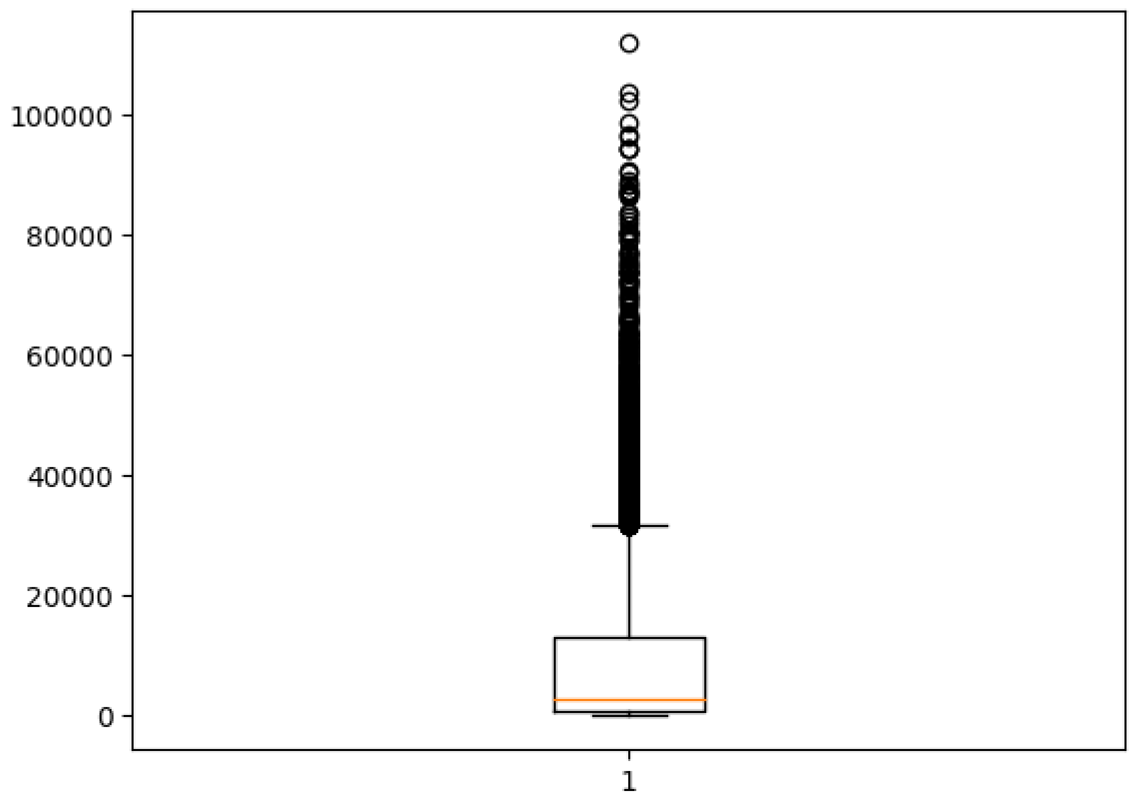
```
6638    20
6664    20
6737    20
6812    20
6818    21
6892     4
7011     4
7017    20
7021    20
7053    20
7080    20
7153    20
7251    21
7252    21
7409     2
7422    20
7548     4
7616    20
7654    21
7767    20
7864    20
7951     3
7984     4
8023    20
8086    20
8090     4
8128    20
8227     4
8343     4
8366     1
8619    20
8631     3
8672     4
8711     3
8775     4
8791     3
8814    20
8838     3
8949     2
9077     3
9081    20
9122    20
9249     2
9332    20
9335    21
9340     3
9343     4
9439    20
9476    22
9639    20
9948    20
Name: Email, dtype: int64
```

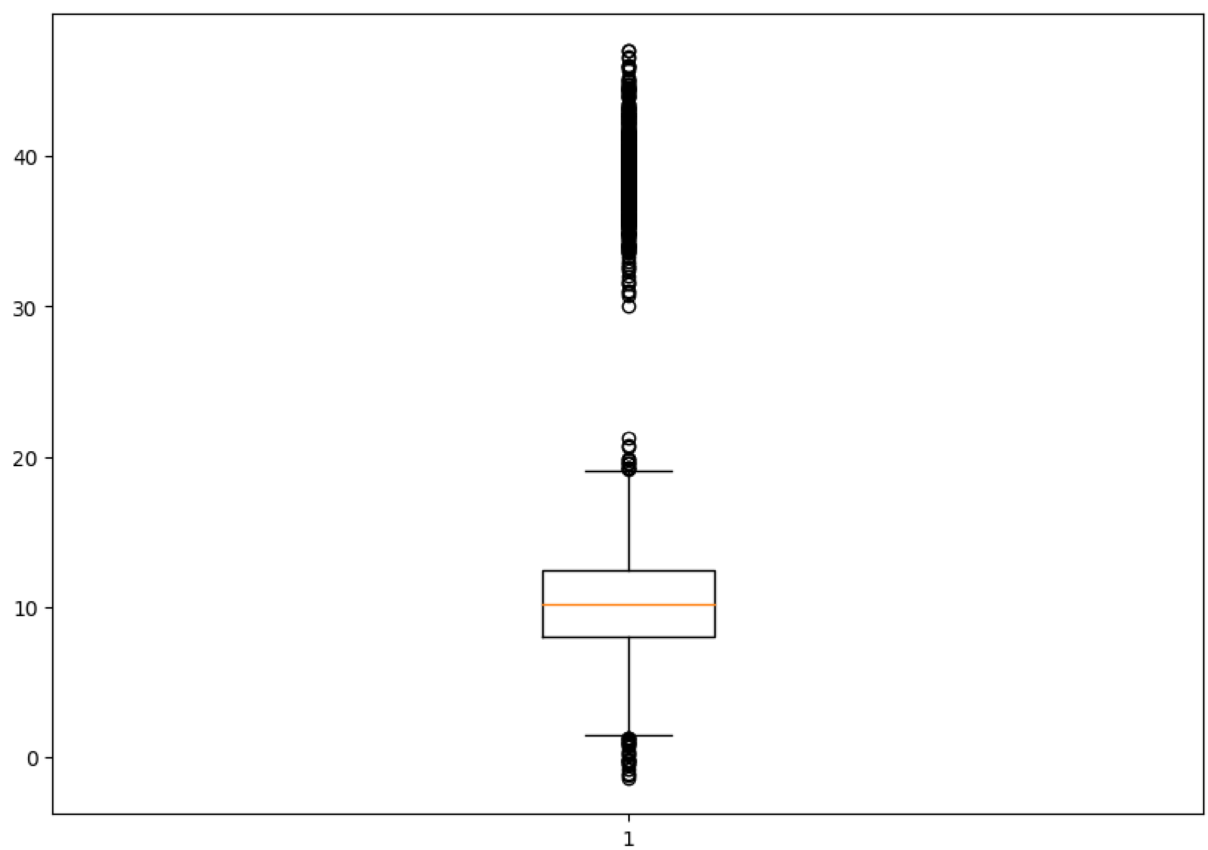
```
2.0 5.0 -3.0
```

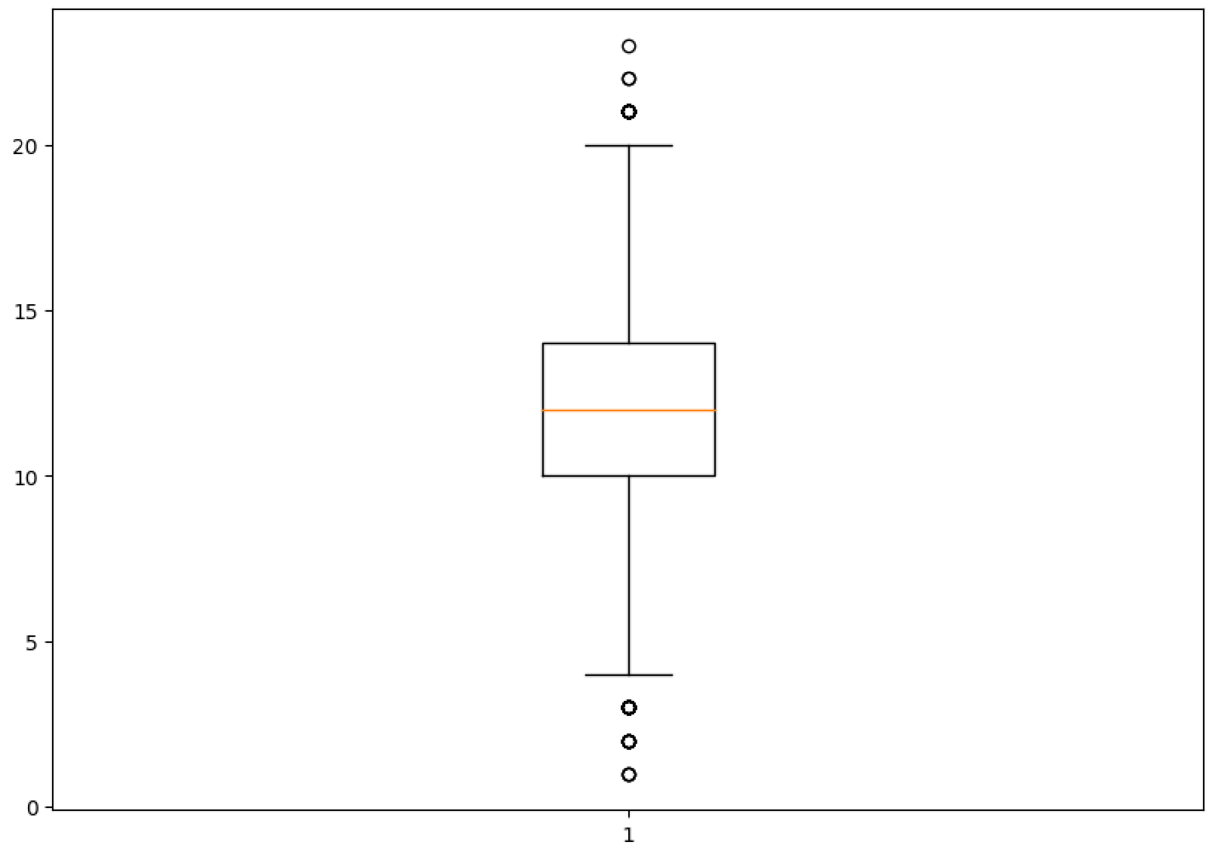
```
---------------------------------------------------------------------
Contacts
---------------------------------------------------------------------
The following are the outliers in the boxplot
188     5
427     6
1926    5
2194    5
2469    5
2571    5
3037    5
3204    5
3904    5
4011    5
4092    5
4297    5
4674    6
4812    7
5138    5
5350    5
5841    6
6285    5
7238    5
7747    7
8039    5
8383    5
9026    5
9173    5
9381    7
9485    5
9714    6
9751    6
Name: Contacts, dtype: int64


1.0 2.5 -1.5
---------------------------------------------------------------------
Yearly_equip_failure
---------------------------------------------------------------------
The following are the outliers in the boxplot
9       3
21      3
172     3
593     3
622     3
698     3
711     3
858     3
1109    3
1117    4
1190    3
1229    4
1307    3
1379    3
1478    3
```

1592	3
1635	3
1842	3
1940	3
2274	3
2336	3
2375	3
2663	3
2819	3
2939	3
3023	3
3238	3
3270	3
3325	3
3451	3
3552	3
3602	3
3900	3
3937	3
3996	3
4269	3
4343	3
4350	3
4384	3
4473	3
4587	3
4687	3
4696	3
5058	3
5076	3
5157	3
5167	4
5224	3
5472	6
5575	3
5769	3
5892	3
6051	3
6099	3
6132	3
6144	3
6197	3
6307	3
6346	4
6367	3
6441	3
6532	3
6587	3
6948	3
7019	3
7112	3
7186	3
7223	3
7279	3
7332	3
7348	3

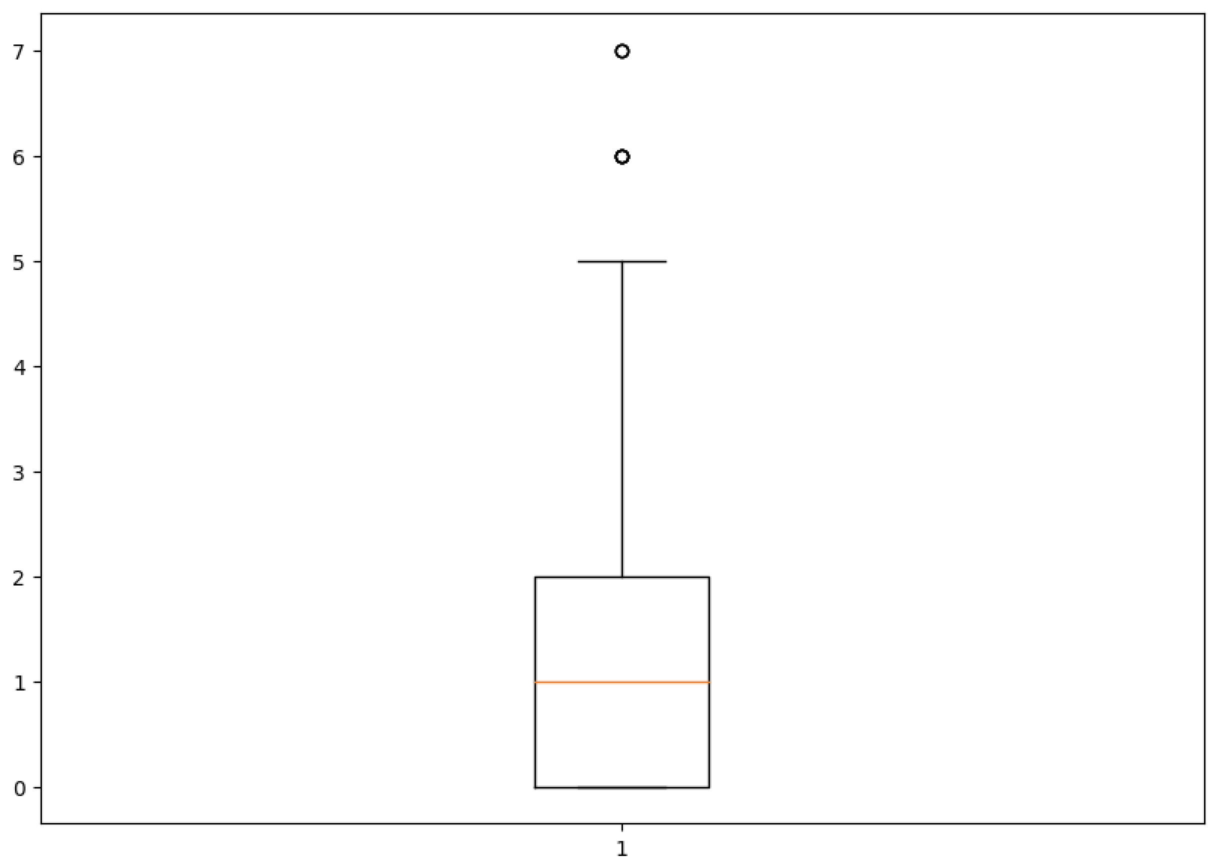
```
7351    3
7455    3
7575    3
7646    3
7891    3
8055    3
8063    3
8282    3
8314    3
8492    3
8981    3
8992    3
9109    3
9176    3
9342    3
9387    4
9423    3
9584    3
9624    4
9675    3
9764    4
9770    3
9968    3
Name: Yearly_equip_failure, dtype: int64


62.706362825000014 297.8369855125 47.01153421249997
---------------------------------------------------------------------
MonthlyCharge
---------------------------------------------------------------------
The following are the outliers in the boxplot
799     299.206164
928     307.528124
1431    298.173023
3747    315.878600
4701    306.268000
Name: MonthlyCharge, dtype: float64
```


100000
80000
60000
40000
20000
0
1

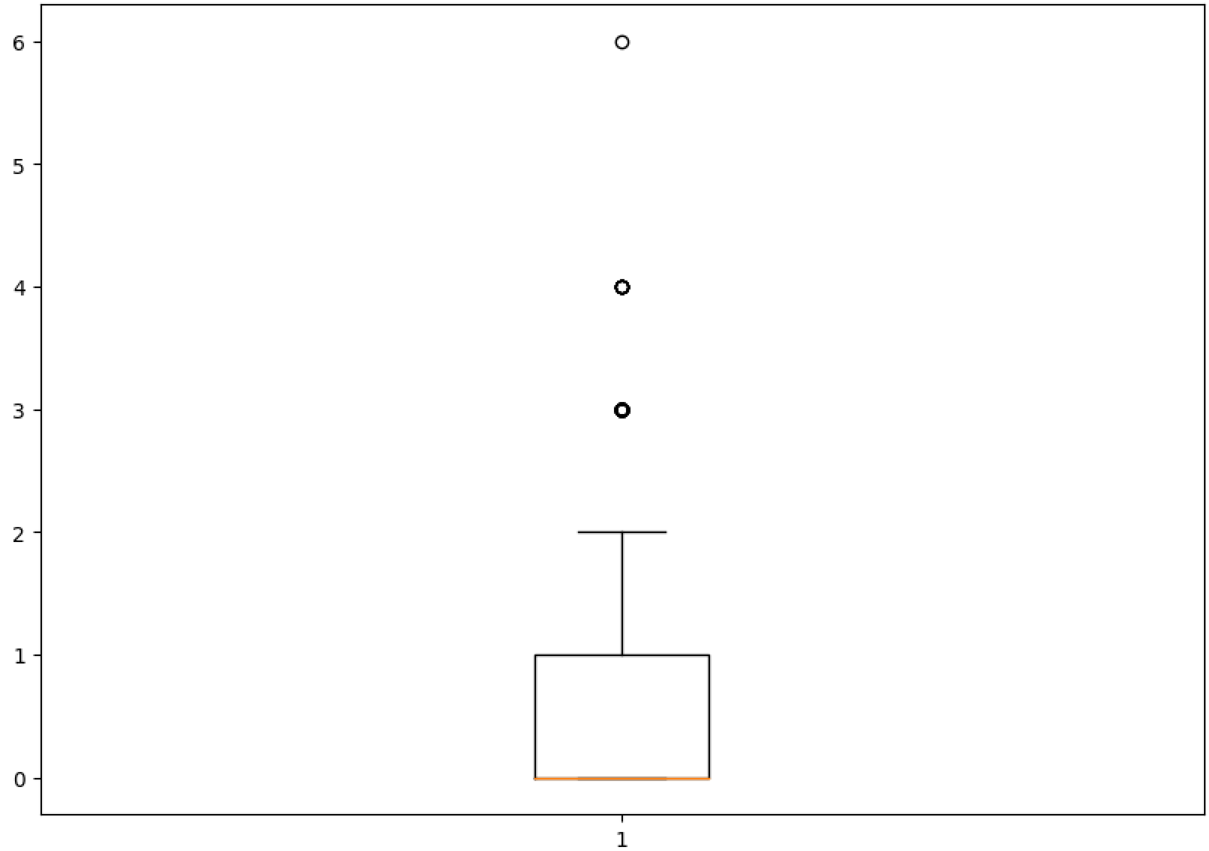

40
30
20
10
0
1

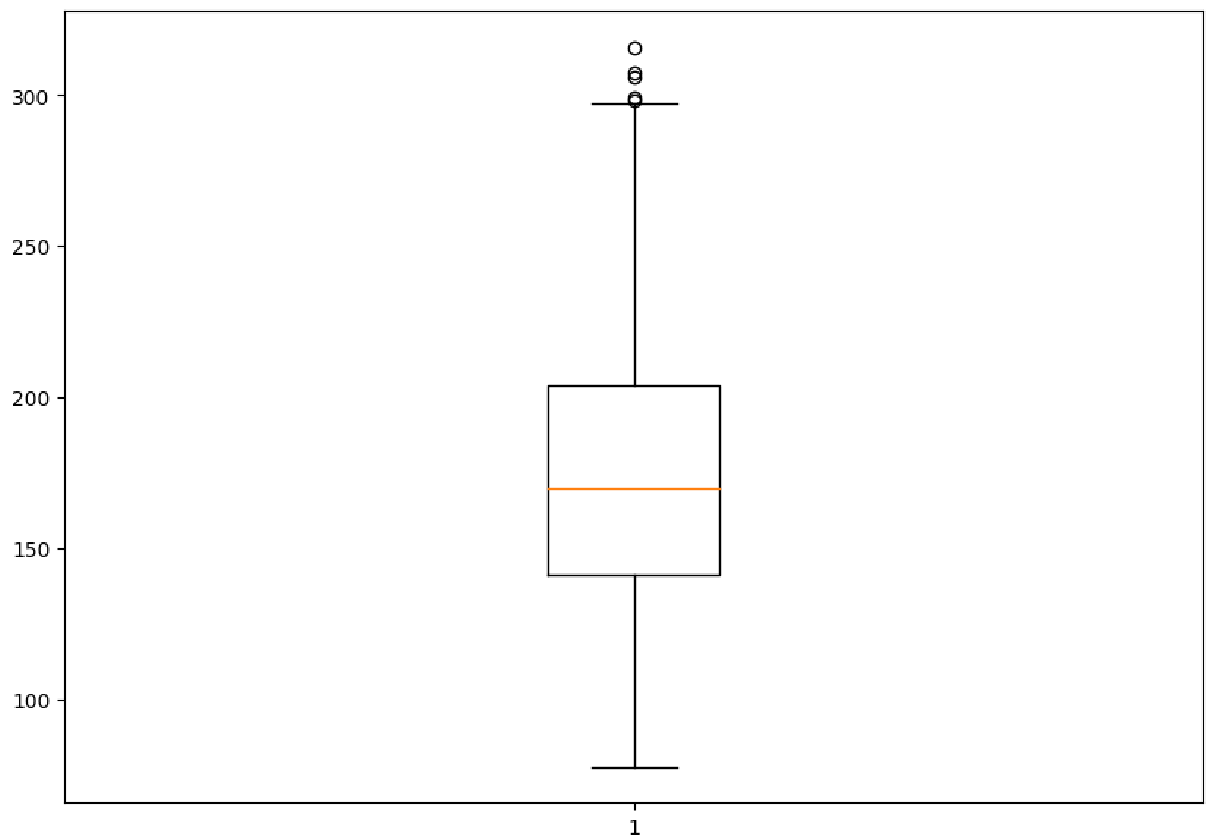


20
15
10
5
0
1



7
6
5
4
3
2
1
0
1






```
<Figure size 1000x700 with 0 Axes>
```

Part III. Data Cleaning

Cleaning Steps:

1. Handle missing numerical values by using mean imputation
2. Correct data types
3. Remove outliers
4. Handle missing categorical values by assigning their respective modes

In [9]:
```
dfx = df.copy()
```

In [10]:
```
dfx.head()
```

Out[10]:

	CaseOrder	Customer_id	Interaction	City	State	County	Zip	Lat	
1	1	K409198	aa90260b-4141-4a24-8e36-b04ce1f4f77b	Point Baker	AK	Prince of Wales-Hyder	99927	56.25100	-13
2	2	S120509	fb76459f-c047-4a9d-8af9-e0f7d4ac2524	West Branch	MI	Ogemaw	48661	44.32893	-8
3	3	K191035	344d114c-3736-4be5-98f7-c72c281e2d35	Yamhill	OR	Yamhill	97148	45.35589	-12
4	4	D90850	abfa2b40-2d43-4994-b15a-989b8c79e311	Del Mar	CA	San Diego	92014	32.96687	-11
5	5	K662701	68a861fd-0d20-4e51-a587-8a90407ee574	Needville	TX	Fort Bend	77461	29.38012	-9

Handle missing numerical values

In [11]:
```
show_missing(dfx)
```

Out[11]:

	num_missing	missing_percentage	num_empty	empty_percentage	na
CaseOrder	0	0.00	0	0.0	
Customer_id	0	0.00	0	0.0	
Interaction	0	0.00	0	0.0	
City	0	0.00	0	0.0	
State	0	0.00	0	0.0	
County	0	0.00	0	0.0	
Zip	0	0.00	0	0.0	
Lat	0	0.00	0	0.0	
Lng	0	0.00	0	0.0	
Population	0	0.00	0	0.0	
Area	0	0.00	0	0.0	
Timezone	0	0.00	0	0.0	
Job	0	0.00	0	0.0	
Children	2495	24.95	0	0.0	
Age	2475	24.75	0	0.0	
Education	0	0.00	0	0.0	
Employment	0	0.00	0	0.0	
Income	2490	24.90	0	0.0	
Marital	0	0.00	0	0.0	
Gender	0	0.00	0	0.0	
Churn	0	0.00	0	0.0	
Outage_sec_perweek	0	0.00	0	0.0	
Email	0	0.00	0	0.0	
Contacts	0	0.00	0	0.0	
Yearly_equip_failure	0	0.00	0	0.0	
Techie	2477	24.77	0	0.0	
Contract	0	0.00	0	0.0	
Port_modem	0	0.00	0	0.0	
Tablet	0	0.00	0	0.0	
InternetService	2129	21.29	0	0.0	

	num_missing	missing_percentage	num_empty	empty_percentage	na
Phone	1026	10.26	0	0.0	
Multiple	0	0.00	0	0.0	
OnlineSecurity	0	0.00	0	0.0	
OnlineBackup	0	0.00	0	0.0	
DeviceProtection	0	0.00	0	0.0	
TechSupport	991	9.91	0	0.0	
StreamingTV	0	0.00	0	0.0	
StreamingMovies	0	0.00	0	0.0	
PaperlessBilling	0	0.00	0	0.0	
PaymentMethod	0	0.00	0	0.0	
Tenure	931	9.31	0	0.0	
MonthlyCharge	0	0.00	0	0.0	
Bandwidth_GB_Year	1021	10.21	0	0.0	
item1	0	0.00	0	0.0	
item2	0	0.00	0	0.0	
item3	0	0.00	0	0.0	
item4	0	0.00	0	0.0	
item5	0	0.00	0	0.0	
item6	0	0.00	0	0.0	
item7	0	0.00	0	0.0	
item8	0	0.00	0	0.0	

In [12]:

```
# assemble list of column names that have missing values (numerical)
missing_columns_num = ['Children',
                       'Age',
                       'Income',
                       'Tenure',
                       'Bandwidth_GB_Year'
                      ]
```

In [13]:

```
def impute_mean(df, column):
    """
    Takes a dataframe and column name and returns
    a dataframe with imputed values
    """
    mean_imputer = SimpleImputer(strategy='mean')
```

```
    df[column] = mean_imputer.fit_transform(df[column].values.reshape(-1,1))
    return df

# loop over missing columns
for col in missing_columns_num:
    dfx = impute_mean(dfx, col)
```

In [14]:

```
show_missing(dfx)
```

Out[14]:

	num_missing	missing_percentage	num_empty	empty_percentage	na
CaseOrder	0	0.00	0	0.0	
Customer_id	0	0.00	0	0.0	
Interaction	0	0.00	0	0.0	
City	0	0.00	0	0.0	
State	0	0.00	0	0.0	
County	0	0.00	0	0.0	
Zip	0	0.00	0	0.0	
Lat	0	0.00	0	0.0	
Lng	0	0.00	0	0.0	
Population	0	0.00	0	0.0	
Area	0	0.00	0	0.0	
Timezone	0	0.00	0	0.0	
Job	0	0.00	0	0.0	
Children	0	0.00	0	0.0	
Age	0	0.00	0	0.0	
Education	0	0.00	0	0.0	
Employment	0	0.00	0	0.0	
Income	0	0.00	0	0.0	
Marital	0	0.00	0	0.0	
Gender	0	0.00	0	0.0	
Churn	0	0.00	0	0.0	
Outage_sec_perweek	0	0.00	0	0.0	
Email	0	0.00	0	0.0	
Contacts	0	0.00	0	0.0	
Yearly_equip_failure	0	0.00	0	0.0	
Techie	2477	24.77	0	0.0	
Contract	0	0.00	0	0.0	
Port_modem	0	0.00	0	0.0	
Tablet	0	0.00	0	0.0	
InternetService	2129	21.29	0	0.0	

	num_missing	missing_percentage	num_empty	empty_percentage	na
Phone	1026	10.26	0	0.0	
Multiple	0	0.00	0	0.0	
OnlineSecurity	0	0.00	0	0.0	
OnlineBackup	0	0.00	0	0.0	
DeviceProtection	0	0.00	0	0.0	
TechSupport	991	9.91	0	0.0	
StreamingTV	0	0.00	0	0.0	
StreamingMovies	0	0.00	0	0.0	
PaperlessBilling	0	0.00	0	0.0	
PaymentMethod	0	0.00	0	0.0	
Tenure	0	0.00	0	0.0	
MonthlyCharge	0	0.00	0	0.0	
Bandwidth_GB_Year	0	0.00	0	0.0	
item1	0	0.00	0	0.0	
item2	0	0.00	0	0.0	
item3	0	0.00	0	0.0	
item4	0	0.00	0	0.0	
item5	0	0.00	0	0.0	
item6	0	0.00	0	0.0	
item7	0	0.00	0	0.0	
item8	0	0.00	0	0.0	

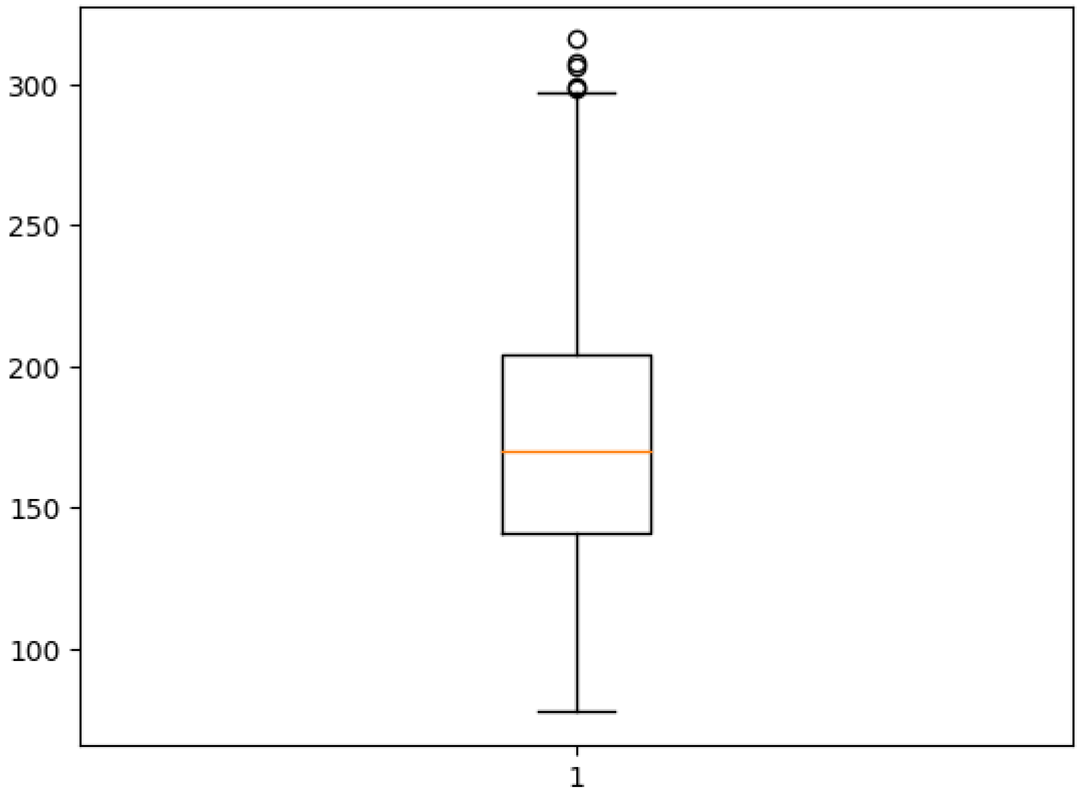
Correct datatypes

In [15]:
```
# cast column values to their correct data types
dfx['Zip'] = dfx['Zip'].astype(str)
dfx['Children'] = dfx['Children'].astype('int64')
dfx['Age'] = dfx['Age'].astype('int64')
```

Remove outliers

In [16]:
```
plt.boxplot(dfx['MonthlyCharge'])
fig = plt.figure(figsize =(10, 7))
```

Out[16]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x1c4840804c0>,
  <matplotlib.lines.Line2D at 0x1c484080760>],
 'caps': [<matplotlib.lines.Line2D at 0x1c4840808e0>,
  <matplotlib.lines.Line2D at 0x1c484080b80>],
 'boxes': [<matplotlib.lines.Line2D at 0x1c484080220>],
 'medians': [<matplotlib.lines.Line2D at 0x1c484080e20>],
 'fliers': [<matplotlib.lines.Line2D at 0x1c48408f100>],
 'means': []}
```




```
<Figure size 1000x700 with 0 Axes>
```

In [17]:
```
# finding the 1st quartile
q1 = np.quantile(dfx['MonthlyCharge'], 0.25)

# finding the 3rd quartile
q3 = np.quantile(dfx['MonthlyCharge'], 0.75)
med = np.median(dfx['MonthlyCharge'])

# finding the iqr region
iqr = q3-q1

# finding upper and lower whiskers
upper_bound = q3+(1.5*iqr)
lower_bound = q1-(1.5*iqr)
print(iqr, upper_bound, lower_bound)

outliers = dfx['MonthlyCharge'][(dfx['MonthlyCharge'] <= lower_bound) | (dfx['Month
print('----------------------------------------------------------------------')
print('The following are the outliers in the boxplot\n{}'.format(outliers))
print('\n\n')
```

```
62.706362825000014 297.8369855125 47.01153421249997
---------------------------------------------------------------------
The following are the outliers in the boxplot
799     299.206164
928     307.528124
1431    298.173023
3747    315.878600
4701    306.268000
Name: MonthlyCharge, dtype: float64
```

In [18]:
```python
dfx.shape

# filter only rows with values below the upper bound and above the lower_bound
dfx = dfx[(dfx["MonthlyCharge"] < upper_bound) & (dfx["MonthlyCharge"] > lower_boun

dfx.shape
```

Out[18]: (10000, 51)

Out[18]: (9995, 51)

Handle missing categorical values

In [19]:
```python
show_missing(dfx)
```

Out[19]:

	num_missing	missing_percentage	num_empty	empty_percentage	na
CaseOrder	0	0.000000	0	0.0	
Customer_id	0	0.000000	0	0.0	
Interaction	0	0.000000	0	0.0	
City	0	0.000000	0	0.0	
State	0	0.000000	0	0.0	
County	0	0.000000	0	0.0	
Zip	0	0.000000	0	0.0	
Lat	0	0.000000	0	0.0	
Lng	0	0.000000	0	0.0	
Population	0	0.000000	0	0.0	
Area	0	0.000000	0	0.0	
Timezone	0	0.000000	0	0.0	
Job	0	0.000000	0	0.0	
Children	0	0.000000	0	0.0	
Age	0	0.000000	0	0.0	
Education	0	0.000000	0	0.0	
Employment	0	0.000000	0	0.0	
Income	0	0.000000	0	0.0	
Marital	0	0.000000	0	0.0	
Gender	0	0.000000	0	0.0	
Churn	0	0.000000	0	0.0	
Outage_sec_perweek	0	0.000000	0	0.0	
Email	0	0.000000	0	0.0	
Contacts	0	0.000000	0	0.0	
Yearly_equip_failure	0	0.000000	0	0.0	
Techie	2477	24.782391	0	0.0	
Contract	0	0.000000	0	0.0	
Port_modem	0	0.000000	0	0.0	
Tablet	0	0.000000	0	0.0	
InternetService	2129	21.300650	0	0.0	

	num_missing	missing_percentage	num_empty	empty_percentage	na
Phone	1025	10.255128	0	0.0	
Multiple	0	0.000000	0	0.0	
OnlineSecurity	0	0.000000	0	0.0	
OnlineBackup	0	0.000000	0	0.0	
DeviceProtection	0	0.000000	0	0.0	
TechSupport	990	9.904952	0	0.0	
StreamingTV	0	0.000000	0	0.0	
StreamingMovies	0	0.000000	0	0.0	
PaperlessBilling	0	0.000000	0	0.0	
PaymentMethod	0	0.000000	0	0.0	
Tenure	0	0.000000	0	0.0	
MonthlyCharge	0	0.000000	0	0.0	
Bandwidth_GB_Year	0	0.000000	0	0.0	
item1	0	0.000000	0	0.0	
item2	0	0.000000	0	0.0	
item3	0	0.000000	0	0.0	
item4	0	0.000000	0	0.0	
item5	0	0.000000	0	0.0	
item6	0	0.000000	0	0.0	
item7	0	0.000000	0	0.0	
item8	0	0.000000	0	0.0	

In [20]:

```
def get_values(df, columns):
    """
    Take a dataframe and a list of columns and
    returns the value counts for the columns.
    """
    for column in columns:
        print(column)
        print('====================================')
        print(df[column].value_counts(dropna=False))
        print('\n')

def show_values(df, param):
    if param == 'all':
        get_values(df, df.columns)
    else:
```

```
        get_values(df, param)

show_values(df, ['Techie', 'InternetService', 'Phone', 'TechSupport'])
```

```
Techie
=====================================
Techie
No     6266
NaN    2477
Yes    1257
Name: count, dtype: int64


InternetService
=====================================
InternetService
Fiber Optic    4408
DSL            3463
NaN            2129
Name: count, dtype: int64


Phone
=====================================
Phone
Yes    8128
NaN    1026
No      846
Name: count, dtype: int64


TechSupport
=====================================
TechSupport
No     5635
Yes    3374
NaN     991
Name: count, dtype: int64
```

In [21]:

```
# fill missing values with the most frequent values of the column (mode)
dfx =  dfx.fillna(value={'Techie':'No',
                         'InternetService':'Fiber Optic',
                         'Phone':'Yes',
                         'TechSupport':'No'
                        })
```

In [22]:

```
show_missing(dfx)
```

Out[22]:

	num_missing	missing_percentage	num_empty	empty_percentage	na
CaseOrder	0	0.0	0	0.0	
Customer_id	0	0.0	0	0.0	
Interaction	0	0.0	0	0.0	
City	0	0.0	0	0.0	
State	0	0.0	0	0.0	
County	0	0.0	0	0.0	
Zip	0	0.0	0	0.0	
Lat	0	0.0	0	0.0	
Lng	0	0.0	0	0.0	
Population	0	0.0	0	0.0	
Area	0	0.0	0	0.0	
Timezone	0	0.0	0	0.0	
Job	0	0.0	0	0.0	
Children	0	0.0	0	0.0	
Age	0	0.0	0	0.0	
Education	0	0.0	0	0.0	
Employment	0	0.0	0	0.0	
Income	0	0.0	0	0.0	
Marital	0	0.0	0	0.0	
Gender	0	0.0	0	0.0	
Churn	0	0.0	0	0.0	
Outage_sec_perweek	0	0.0	0	0.0	
Email	0	0.0	0	0.0	
Contacts	0	0.0	0	0.0	
Yearly_equip_failure	0	0.0	0	0.0	
Techie	0	0.0	0	0.0	
Contract	0	0.0	0	0.0	
Port_modem	0	0.0	0	0.0	
Tablet	0	0.0	0	0.0	
InternetService	0	0.0	0	0.0	

	num_missing	missing_percentage	num_empty	empty_percentage	na
Phone	0	0.0	0	0.0	
Multiple	0	0.0	0	0.0	
OnlineSecurity	0	0.0	0	0.0	
OnlineBackup	0	0.0	0	0.0	
DeviceProtection	0	0.0	0	0.0	
TechSupport	0	0.0	0	0.0	
StreamingTV	0	0.0	0	0.0	
StreamingMovies	0	0.0	0	0.0	
PaperlessBilling	0	0.0	0	0.0	
PaymentMethod	0	0.0	0	0.0	
Tenure	0	0.0	0	0.0	
MonthlyCharge	0	0.0	0	0.0	
Bandwidth_GB_Year	0	0.0	0	0.0	
item1	0	0.0	0	0.0	
item2	0	0.0	0	0.0	
item3	0	0.0	0	0.0	
item4	0	0.0	0	0.0	
item5	0	0.0	0	0.0	
item6	0	0.0	0	0.0	
item7	0	0.0	0	0.0	
item8	0	0.0	0	0.0	

In [23]:

```
# Select rows that are duplicated based on all columns. Any records after the first
dup = dfx[dfx.duplicated()]

# Find out how many rows are duplicated
dup.shape
```

Out[23]: (0, 51)

In [24]:

```
dfx.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9995 entries, 1 to 10000
Data columns (total 51 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CaseOrder             9995 non-null   int64  
 1   Customer_id           9995 non-null   object 
 2   Interaction           9995 non-null   object 
 3   City                  9995 non-null   object 
 4   State                 9995 non-null   object 
 5   County                9995 non-null   object 
 6   Zip                   9995 non-null   object 
 7   Lat                   9995 non-null   float64
 8   Lng                   9995 non-null   float64
 9   Population            9995 non-null   int64  
 10  Area                  9995 non-null   object 
 11  Timezone              9995 non-null   object 
 12  Job                   9995 non-null   object 
 13  Children              9995 non-null   int64  
 14  Age                   9995 non-null   int64  
 15  Education             9995 non-null   object 
 16  Employment            9995 non-null   object 
 17  Income                9995 non-null   float64
 18  Marital               9995 non-null   object 
 19  Gender                9995 non-null   object 
 20  Churn                 9995 non-null   object 
 21  Outage_sec_perweek    9995 non-null   float64
 22  Email                 9995 non-null   int64  
 23  Contacts              9995 non-null   int64  
 24  Yearly_equip_failure  9995 non-null   int64  
 25  Techie                9995 non-null   object 
 26  Contract              9995 non-null   object 
 27  Port_modem            9995 non-null   object 
 28  Tablet                9995 non-null   object 
 29  InternetService       9995 non-null   object 
 30  Phone                 9995 non-null   object 
 31  Multiple              9995 non-null   object 
 32  OnlineSecurity        9995 non-null   object 
 33  OnlineBackup          9995 non-null   object 
 34  DeviceProtection      9995 non-null   object 
 35  TechSupport           9995 non-null   object 
 36  StreamingTV           9995 non-null   object 
 37  StreamingMovies       9995 non-null   object 
 38  PaperlessBilling      9995 non-null   object 
 39  PaymentMethod         9995 non-null   object 
 40  Tenure                9995 non-null   float64
 41  MonthlyCharge         9995 non-null   float64
 42  Bandwidth_GB_Year     9995 non-null   float64
 43  item1                 9995 non-null   int64  
 44  item2                 9995 non-null   int64  
 45  item3                 9995 non-null   int64  
 46  item4                 9995 non-null   int64  
 47  item5                 9995 non-null   int64  
 48  item6                 9995 non-null   int64  
 49  item7                 9995 non-null   int64  
 50  item8                 9995 non-null   int64  
```

```
dtypes: float64(7), int64(15), object(29)
memory usage: 4.0+ MB
```

Cleaned dataset

In [25]:
```python
dfx.head()
dfx.tail()

dfx.to_csv('churn_cleaned_data_notebook.csv', index=False)
```

Out[25]:

	CaseOrder	Customer_id	Interaction	City	State	County	Zip	Lat	
1	1	K409198	aa90260b-4141-4a24-8e36-b04ce1f4f77b	Point Baker	AK	Prince of Wales-Hyder	99927	56.25100	-13
2	2	S120509	fb76459f-c047-4a9d-8af9-e0f7d4ac2524	West Branch	MI	Ogemaw	48661	44.32893	-8
3	3	K191035	344d114c-3736-4be5-98f7-c72c281e2d35	Yamhill	OR	Yamhill	97148	45.35589	-12
4	4	D90850	abfa2b40-2d43-4994-b15a-989b8c79e311	Del Mar	CA	San Diego	92014	32.96687	-11
5	5	K662701	68a861fd-0d20-4e51-a587-8a90407ee574	Needville	TX	Fort Bend	77461	29.38012	-9

Out[25]:

	CaseOrder	Customer_id	Interaction	City	State	County	Zip	
9996	9996	M324793	45deb5a2-ae04-4518-bf0b-c82db8dbe4a4	Mount Holly	VT	Rutland	5758	43.4
9997	9997	D861732	6e96b921-0c09-4993-bbda-a1ac6411061a	Clarksville	TN	Montgomery	37042	36.5
9998	9998	I243405	e8307ddf-9a01-4fff-bc59-4742e03fd24f	Mobeetie	TX	Wheeler	79061	35.5
9999	9999	I641617	3775ccfc-0052-4107-81ae-9657f81ecdf3	Carrollton	GA	Carroll	30117	33.5
10000	10000	T38070	9de5fb6e-bd33-4995-aec8-f01d0172a499	Clarkesville	GA	Habersham	30523	34.7

D1. Cleaning Findings

1. Data set has no duplicates.
2. Data set has missing values both numerical and categorical.
3. A few columns had incorrect data types.
4. Outliers exist in several columns but only "MonthlyCharge" makes sense to remove.

D2. Justification of Mitigation Efforts

1. The data set has no duplicates so no duplicates were removed.
2. The data set has several columns with missing data or nulls and NaNs. For the numerical values, the missing values were imputed by mean. For the categorical, the missing values were assigned their respective modes.
3. Incorrect data types had to be dealt with manually by casting the column values to their correct data type. "Zip" was cast as a string. "Children" and "Age" were cast as integers.
4. While the data set has several columns with outliers, I have decided to only remove the outliers in "MonthlyCharge" because it is the only column that retains its truest form when the outliers were removed. Removing the outliers in other columns would skew data distribution.

D3. Summary of the Outcomes

1. The dataframe was unchanged.
2. The dataframe retained its shape.
3. The dataframe retained its shape.
4. The dataframe decreased by five in the number of rows. The number of columns remained the same.

In the end, there were 9,995 rows and 51 columns.

D4. Mitigation Code (Executable File)

Filename: clean_churn.py

```
#!/usr/bin/env python

""" WGU D206 Data Cleaning Performance Assessment """

import sys

# setting the random seed for reproducibility
import random
random.seed(493)

import pandas as pd # for manipulating dataframes
import numpy as np # for numerical operations
import matplotlib.pyplot as plt # for visualization
from sklearn.impute import SimpleImputer # for handling missing
values

def impute_mean(df, column):
    """
    Takes a dataframe and column name and returns
    a dataframe with imputed values
    """
    mean_imputer = SimpleImputer(strategy='mean')
    df[column] =
mean_imputer.fit_transform(df[column].values.reshape(-1,1))
    return df

def main():
    """Main entry point for the script."""

    # Read a csv file
    df = pd.read_csv('churn_raw_data.csv', index_col=0)
    dfx = df.copy()

    # assemble list of column names that have missing values
(numerical)
    missing_columns_num = ['Children',
                    'Age',
                    'Income',
```

```
                    'Tenure',
                    'Bandwidth_GB_Year'
                    ]

    def impute_mean(df, column):
        """
        Takes a dataframe and column name and returns
        a dataframe with imputed values
        """
        mean_imputer = SimpleImputer(strategy='mean')
        df[column] =
mean_imputer.fit_transform(df[column].values.reshape(-1,1))
        return df

    # loop over missing columns
    for col in missing_columns_num:
        dfx = impute_mean(dfx, col)

    # cast column values to their correct data types
    dfx['Zip'] = dfx['Zip'].astype(str)
    dfx['Children'] = dfx['Children'].astype('int64')
    dfx['Age'] = dfx['Age'].astype('int64')

    # finding the 1st quartile
    q1 = np.quantile(dfx['MonthlyCharge'], 0.25)

    # finding the 3rd quartile
    q3 = np.quantile(dfx['MonthlyCharge'], 0.75)
    med = np.median(dfx['MonthlyCharge'])

    # finding the iqr region
    iqr = q3-q1

    # finding upper and lower whiskers
    upper_bound = q3+(1.5*iqr)
    lower_bound = q1-(1.5*iqr)

    # filter only rows with values below the upper bound and above
the lower_bound
    dfx = dfx[(dfx["MonthlyCharge"] < upper_bound) &
(dfx["MonthlyCharge"] > lower_bound)]

    # fill missing values with the most frequent values of the
column (mode)
    dfx =  dfx.fillna(value={'Techie':'No',
                            'InternetService':'Fiber Optic',
                            'Phone':'Yes',
                            'TechSupport':'No'
                            })
```

```
        dfx.to_csv('churn_cleaned_data_executable.csv', index=False)
        print('D
    __maintainer__ = "Ednalyn C. De Dios"
    __email__ = "ednalyn.dedios@gmail.com"
    __status__ = "Prototype"data.csv', index=False)

    if __name__ == '__main__':
        sys.exit(main())
```

D5. Clean Data

Filename: churn_cleaned_data_notebook.csv

D6. Limitations

During the data cleaning step of handling missing values, the categorical columns had a significant number of missing values. If the rows that have this missing values were to be dropped, the shape of the dataframe would change dramatically from 10,000 to 4,781 records. Hence, the missing values of the categorical type were filled with the respective mode instead.

This presents a minor limitation in future analysis if these categorical variables were to be examined.

D7. Impact of Limitations

However, since the current research question mainly dealt with only "Churn" and "PaymentMethod," the current mitigation step to handle the missing categorical values was of no consequence to the current research question.

E1. Principal Components

In [26]:
```
dfx.head()
```

Out[26]:

	CaseOrder	Customer_id	Interaction	City	State	County	Zip	Lat	
1	1	K409198	aa90260b-4141-4a24-8e36-b04ce1f4f77b	Point Baker	AK	Prince of Wales-Hyder	99927	56.25100	-13
2	2	S120509	fb76459f-c047-4a9d-8af9-e0f7d4ac2524	West Branch	MI	Ogemaw	48661	44.32893	-8
3	3	K191035	344d114c-3736-4be5-98f7-c72c281e2d35	Yamhill	OR	Yamhill	97148	45.35589	-12
4	4	D90850	abfa2b40-2d43-4994-b15a-989b8c79e311	Del Mar	CA	San Diego	92014	32.96687	-11
5	5	K662701	68a861fd-0d20-4e51-a587-8a90407ee574	Needville	TX	Fort Bend	77461	29.38012	-9

In [27]:

```
data = dfx.copy()
```

In [28]:

```
# create data set of numeric columns
df_num = data.select_dtypes(include='number')
df_num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9995 entries, 1 to 10000
Data columns (total 22 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CaseOrder             9995 non-null   int64  
 1   Lat                   9995 non-null   float64
 2   Lng                   9995 non-null   float64
 3   Population            9995 non-null   int64  
 4   Children              9995 non-null   int64  
 5   Age                   9995 non-null   int64  
 6   Income                9995 non-null   float64
 7   Outage_sec_perweek    9995 non-null   float64
 8   Email                 9995 non-null   int64  
 9   Contacts              9995 non-null   int64  
 10  Yearly_equip_failure  9995 non-null   int64  
 11  Tenure                9995 non-null   float64
 12  MonthlyCharge         9995 non-null   float64
 13  Bandwidth_GB_Year     9995 non-null   float64
 14  item1                 9995 non-null   int64  
 15  item2                 9995 non-null   int64  
 16  item3                 9995 non-null   int64  
 17  item4                 9995 non-null   int64  
 18  item5                 9995 non-null   int64  
 19  item6                 9995 non-null   int64  
 20  item7                 9995 non-null   int64  
 21  item8                 9995 non-null   int64  
dtypes: float64(7), int64(15)
memory usage: 1.8 MB
```

In [29]:

```
# remove irrelevant columns
df_num = df_num.drop(columns=['CaseOrder', 'Lat', 'Lng'])
df_num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9995 entries, 1 to 10000
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Population            9995 non-null   int64  
 1   Children              9995 non-null   int64  
 2   Age                   9995 non-null   int64  
 3   Income                9995 non-null   float64
 4   Outage_sec_perweek    9995 non-null   float64
 5   Email                 9995 non-null   int64  
 6   Contacts              9995 non-null   int64  
 7   Yearly_equip_failure  9995 non-null   int64  
 8   Tenure                9995 non-null   float64
 9   MonthlyCharge         9995 non-null   float64
 10  Bandwidth_GB_Year     9995 non-null   float64
 11  item1                 9995 non-null   int64  
 12  item2                 9995 non-null   int64  
 13  item3                 9995 non-null   int64  
 14  item4                 9995 non-null   int64  
 15  item5                 9995 non-null   int64  
 16  item6                 9995 non-null   int64  
 17  item7                 9995 non-null   int64  
 18  item8                 9995 non-null   int64  
dtypes: float64(5), int64(14)
memory usage: 1.5 MB
```

Normalize numeric dataframe

In [30]:
```
from sklearn.preprocessing import StandardScaler

features = df_num.columns
x = df_num.loc[:, features].values
x = StandardScaler().fit_transform(x) # normalizing the features
```

In [31]:
```
x.shape
```

Out[31]:
```
(9995, 19)
```

In [32]:
```
np.mean(x),np.std(x)
```

Out[32]:
```
(6.921903378827231e-18, 1.0)
```

In [33]:
```
feat_cols = ['feature'+str(i) for i in range(x.shape[1])]
```

In [34]:
```
normalised_df = pd.DataFrame(x,columns=feat_cols)
```

In [35]:
```
normalised_df.head()
```

Out[35]:

	feature0	feature1	feature2	feature3	feature4	feature5	feature6	feature7	f
0	-0.673586	-0.038686	0.821738	-0.463132	-0.638002	-0.666026	-1.005857	0.946201	-1
1	0.047472	-0.574241	-1.455683	-0.742167	0.082128	-0.004960	-1.005857	0.946201	-1
2	-0.417461	1.032422	-0.178106	-0.000267	-0.170522	-0.996559	-1.005857	0.946201	-0
3	0.284199	-0.574241	-0.289199	-0.855272	0.537982	0.986638	1.017803	-0.626081	-0
4	0.110239	-1.109795	1.654940	0.005325	-0.354097	1.317171	1.017803	0.946201	-1

Apply PCA

In [36]:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.85)
pc = pca.fit_transform(x)
pca.n_components_
```

Out[36]:

```
13
```

In [37]:

```
pca_df = pd.DataFrame(data = pc)
pca_df.head()
```

Out[37]:

	0	1	2	3	4	5	6	7	
0	1.934594	-1.419864	1.908546	-0.253047	0.620822	0.948128	1.378774	-0.330418	0.0
1	-0.205834	-1.668628	0.530546	1.430787	0.636938	-0.672177	1.063801	-0.369013	1.0
2	-0.670827	-0.953811	0.263852	-0.181192	1.825431	0.483069	0.721415	0.141026	0.1
3	0.043273	-0.753827	2.225064	-0.361443	-1.311256	-0.854130	0.033033	-0.337480	-0.3
4	1.334634	-1.960552	0.792410	-0.427770	-1.895925	0.894676	0.741967	0.890928	-1.0

Create scree plot

In [38]:

```
PC_values = np.arange(pca.n_components_) + 1
plt.plot(PC_values, pca.explained_variance_ratio_, 'b+-', linewidth=1)
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.show()
```

Out[38]:

```
[<matplotlib.lines.Line2D at 0x1c484d392e0>]
```

Out[38]:

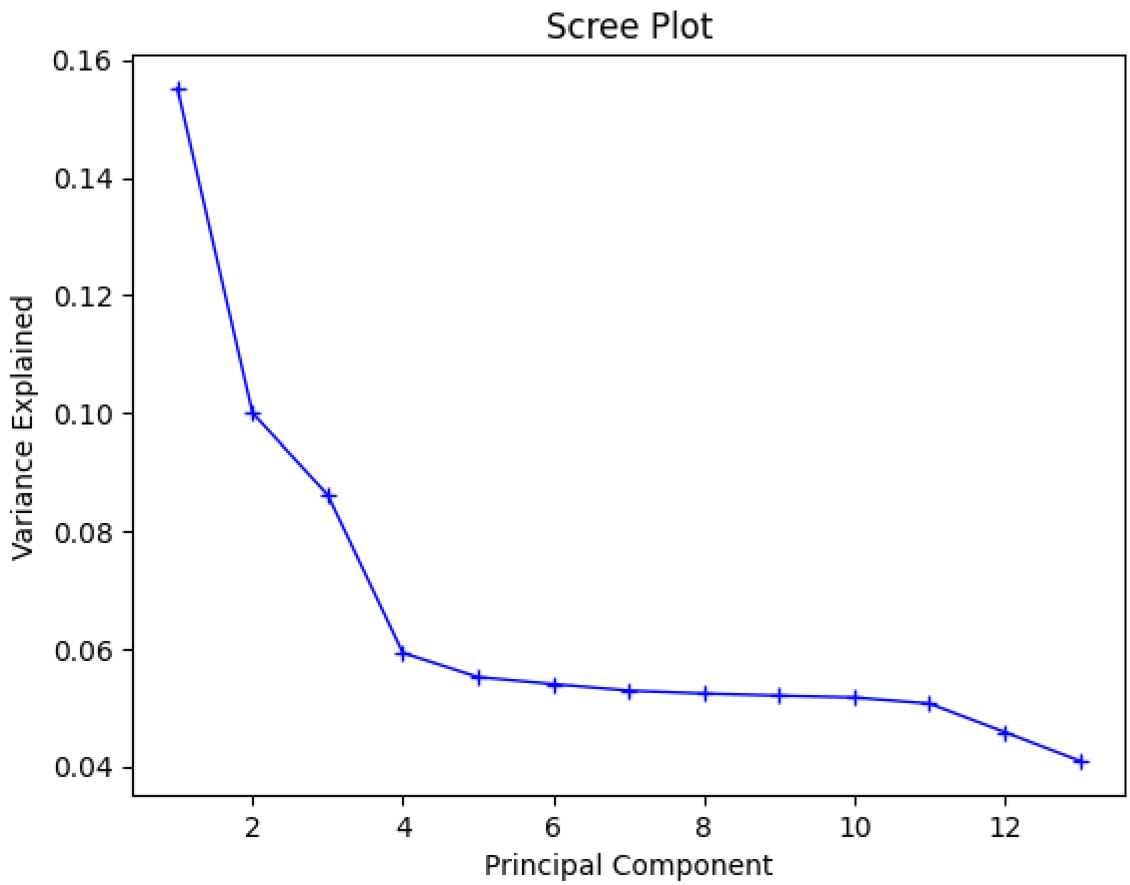
```
Text(0.5, 1.0, 'Scree Plot')
```

Out[38]:

```
Text(0.5, 0, 'Principal Component')
```

Out[38]:

```
Text(0, 0.5, 'Variance Explained')
```



Display Explained Variance Ratios

In [39]:
```
print('Explained variation per principal component: {}'.format(pca.explained_varian
```

```
Explained variation per principal component: [0.15517006 0.10000786 0.0861633  0.059
30058 0.05518868 0.05399561
 0.05288829 0.05238832 0.05203487 0.05171756 0.0506778  0.04584545
 0.04094386]
```

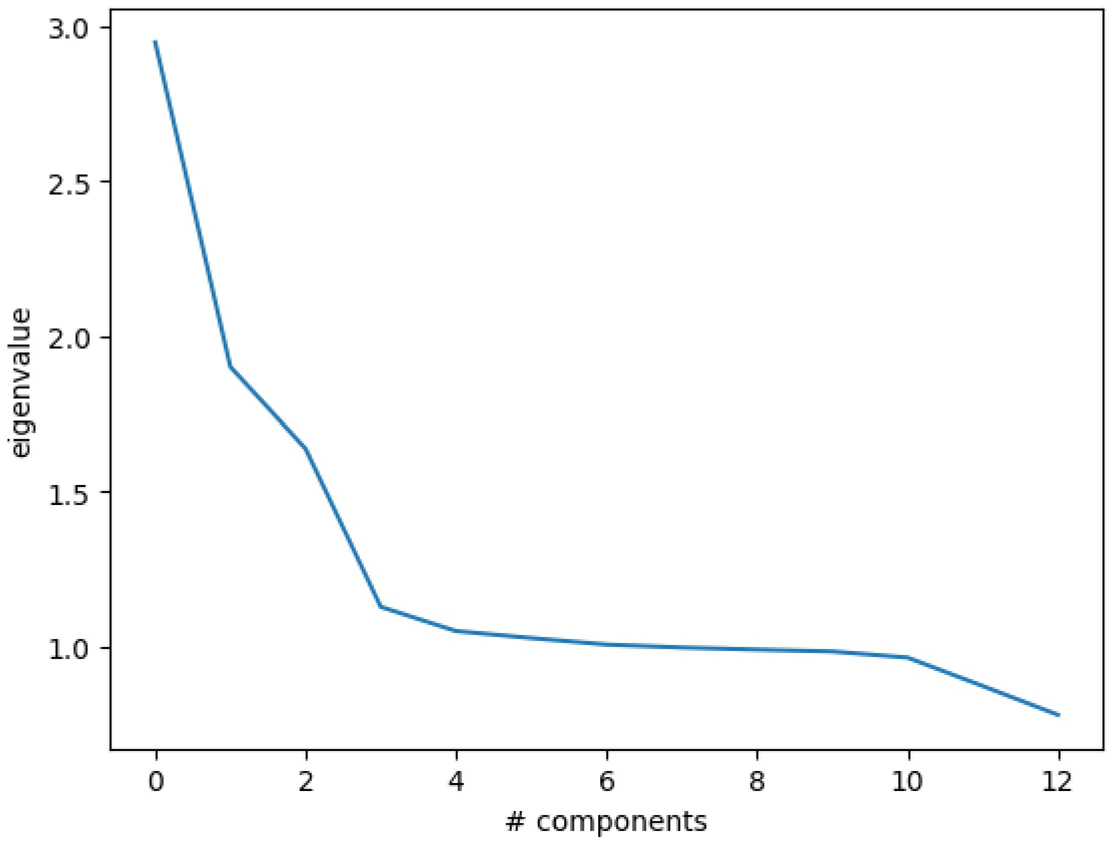
Plot eigenvalues

In [40]:
```
matrix = np.dot(normalised_df.T, normalised_df) / df_num.shape[0]
eigenvalues = [np.dot(eigenvector.T, np.dot(matrix, eigenvector)) for eigenvector i

# plot eigenvalues
plt.plot(eigenvalues)
plt.xlabel('# components')
plt.ylabel('eigenvalue')
plt.show()
```

Out[40]:
```
[<matplotlib.lines.Line2D at 0x1c484b93a90>]
```

Out[40]:
```
Text(0.5, 0, '# components')
```

Out[40]:
```
Text(0, 0.5, 'eigenvalue')
```



Display component values

In [41]:
```
# display list of component values
values = pd.DataFrame(pca.components_.T, index=df_num.columns)
values.round(3)
```

Out[41]:

	0	1	2	3	4	5	6	7	8	
Population	-0.002	0.000	0.015	-0.040	-0.310	-0.387	0.047	0.678	0.438	0
Children	0.001	0.001	0.011	0.008	0.568	-0.207	-0.084	0.202	-0.486	0
Age	0.005	-0.013	-0.017	-0.048	-0.399	0.469	0.176	0.154	-0.357	-0
Income	-0.001	0.008	0.025	0.010	0.210	0.289	-0.716	0.448	0.014	-0
Outage_sec_perweek	-0.013	0.022	-0.047	0.705	0.024	-0.017	-0.015	0.054	0.029	0
Email	0.008	-0.021	-0.004	0.040	-0.301	-0.558	-0.002	0.140	-0.601	-0
Contacts	-0.009	0.004	-0.010	-0.013	-0.446	0.291	-0.253	0.086	-0.255	0
Yearly_equip_failure	-0.008	0.016	0.007	0.076	0.275	0.324	0.615	0.487	-0.092	0
Tenure	-0.011	0.701	-0.070	-0.062	-0.018	-0.005	0.004	0.000	-0.011	-0
MonthlyCharge	-0.000	0.046	-0.024	0.695	-0.099	0.026	-0.034	-0.074	0.013	-0
Bandwidth_GB_Year	-0.013	0.703	-0.073	-0.013	0.004	-0.017	-0.004	-0.003	-0.009	0
item1	0.459	0.032	0.280	0.032	0.006	0.006	0.017	-0.001	-0.002	0
item2	0.434	0.043	0.281	0.018	-0.010	0.018	-0.002	-0.011	-0.010	0
item3	0.401	0.035	0.281	-0.014	-0.006	-0.024	0.021	-0.025	0.006	-0
item4	0.146	-0.049	-0.567	-0.031	-0.001	0.000	0.016	-0.012	0.020	0
item5	-0.176	0.065	0.586	0.025	-0.034	0.016	0.006	-0.018	0.005	-0
item6	0.405	-0.012	-0.183	0.006	-0.003	-0.004	-0.002	0.023	-0.009	0
item7	0.358	-0.003	-0.181	-0.031	0.022	0.012	-0.055	0.008	-0.036	0
item8	0.309	-0.017	-0.132	0.030	-0.021	0.019	0.017	0.022	0.075	-0

E2. Criteria Used

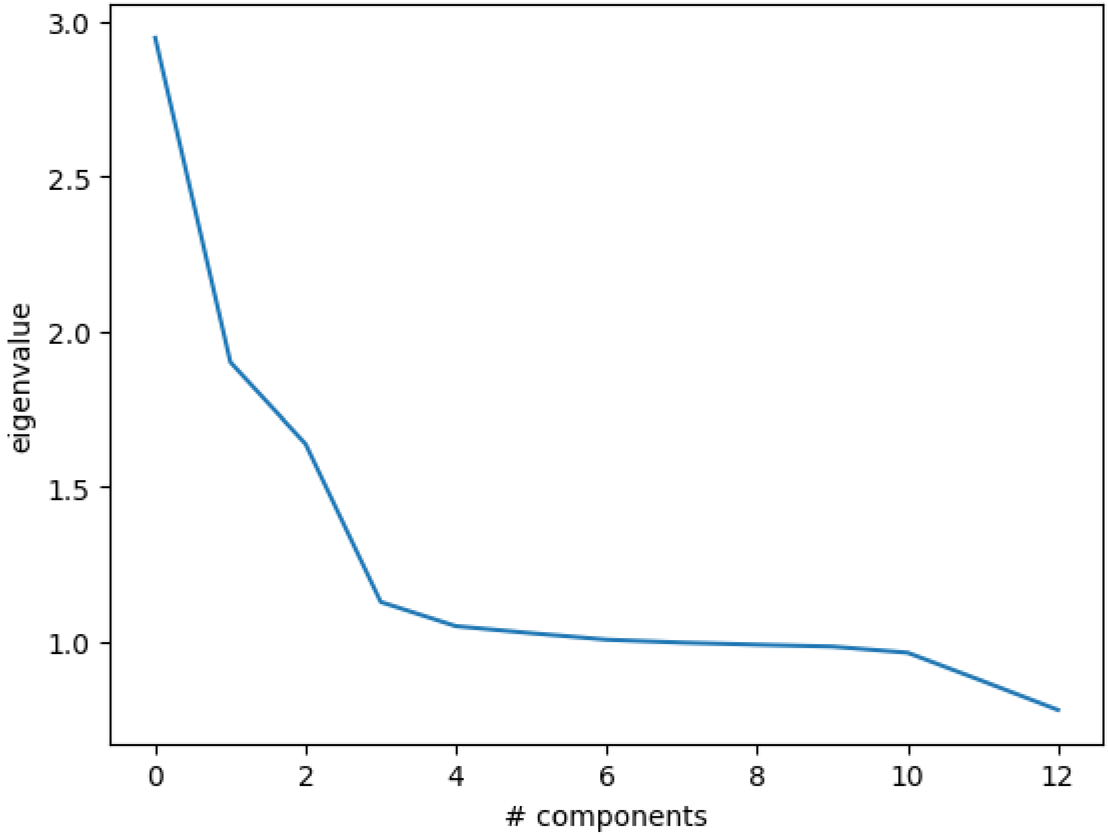
After normalizing and selecting numeric features, I applied Scikit Learn's PCA and chose to keep about 85% of the variance in the original data. This resulted in the selection of 13 components.

In [42]:
```
# plot eigenvalues
plt.plot(eigenvalues)
plt.xlabel('# components')
plt.ylabel('eigenvalue')
plt.show()
```

Out[42]: `[<matplotlib.lines.Line2D at 0x1c484419580>]`

Out[42]: `Text(0.5, 0, '# components')`

```
Out[42]: Text(0, 0.5, 'eigenvalue')
```



E3. Benefits

The table of component values suggests that MonthlyCharge and Outage_sec_perweek are important features. The organization would benefit from this information by mitigating outages and curbing fees.

F. Video

URL: https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=d319fe78-9fdd-4f44-89df-b05700298e30

G. Sources of Third-Party Code

- https://www.datacamp.com/tutorial/principal-component-analysis-in-python
- https://towardsdatascience.com/how-to-select-the-best-number-of-principal-components-for-the-dataset-287e64b14c6d
- https://www.geeksforgeeks.org/find-duplicate-rows-in-a-dataframe-based-on-all-or-selected-columns/
- https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/
- https://www.geeksforgeeks.org/finding-the-outlier-points-from-matplotlib/
- https://towardsdatascience.com/imputing-missing-data-with-simple-and-advanced-techniques-f5c7b157fb87

- https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.htmls

H. Sources

I did not use any other sources to write the text in this document.

In [43]:
```
print('Successful run!')
```

```
Successful run!
```

In [ ]: